



US005638448A

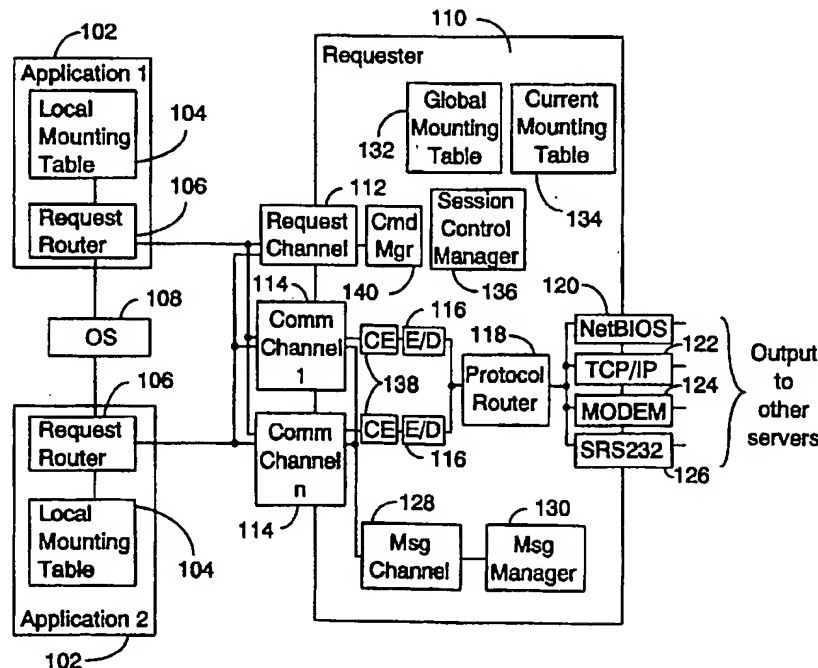
**United States Patent** [19][11] **Patent Number:** **5,638,448****Nguyen**[45] **Date of Patent:** **Jun. 10, 1997****[54] NETWORK WITH SECURE COMMUNICATIONS SESSIONS****[76] Inventor:** Minhtam C. Nguyen, 335 Elan Village  
La., Apt. 217, San Jose, Calif. 95134**[21] Appl. No.:** 583,933**[22] Filed:** Jan. 11, 1996**Related U.S. Application Data****[63]** Continuation-in-part of Ser. No. 547,346, Oct. 24, 1995.**[51] Int. CL<sup>6</sup>** ..... H04L 9/06; H04L 9/32;  
H04L 9/00**[52] U.S. Cl.** ..... 380/29; 380/9; 380/21;  
380/23; 380/25; 380/37; 380/43; 380/49**[58] Field of Search** ..... 380/9, 21, 23,  
380/25, 29, 37, 43, 49, 50, 59**[56] References Cited****U.S. PATENT DOCUMENTS**

4,227,253	10/1980	Ehrsam et al.	375/2
5,060,263	10/1991	Bosen et al.	380/25
5,073,852	12/1991	Siegel et al.	395/700
5,111,504	5/1992	Esseman et al.	380/21
5,136,716	8/1992	Harvey et al.	395/800
5,142,622	8/1992	Owens	395/200
5,220,655	6/1993	Tsutsui	395/325
5,226,172	7/1993	Seymour et al.	395/800
5,239,648	8/1993	Nukui	395/600
5,241,599	8/1993	Bellovin et al.	380/21
5,261,070	11/1993	Ohta	395/425
5,263,165	11/1993	Janis	395/725
5,268,962	12/1993	Abadi et al.	380/21

5,301,247	4/1994	Rasmussen et al.	380/43
5,311,593	5/1994	Carmi	380/23
5,323,146	6/1994	Glaschick	340/825.34
5,349,642	9/1994	Kingdon	380/25
5,369,707	11/1994	Pollendore, III	380/25
5,373,559	12/1994	Kaufman et al.	380/30
5,375,207	12/1994	Blakely et al.	395/200
5,392,357	2/1995	Bulfer et al.	380/33
5,416,842	5/1995	Aziz	380/30
5,418,854	5/1995	Kaufman et al.	380/23
5,559,883	9/1996	Williams	380/25 X

**OTHER PUBLICATIONS**Bruce Schneier, *Applied Cryptography* (second edition),  
New York, NY, John Wiley & Sons, Inc, 1996, pp. 298-301.*Primary Examiner*—Bernarr E. Gregory  
*Attorney, Agent, or Firm*—John C. Smith**[57] ABSTRACT**

A system which uses three way password authentication, and s<sup>3</sup>DES to encrypt different portions of a logon packet with different keys based on the nature of the communications link. Nodes attached to a particular LAN can have one level of security for data transfer within the LAN while data transfers between LANs on a private network can have a second level of security and LANs connected via public networks can have a third level of security. The level of security can optionally be selected by the user. Data transfers between nodes of a network are kept in separate queues to reduce queue search times and enhance performance. Each session maintains its own key dependent s<sup>3</sup>DES S-boxes to enhance security.

**20 Claims, 15 Drawing Sheets**

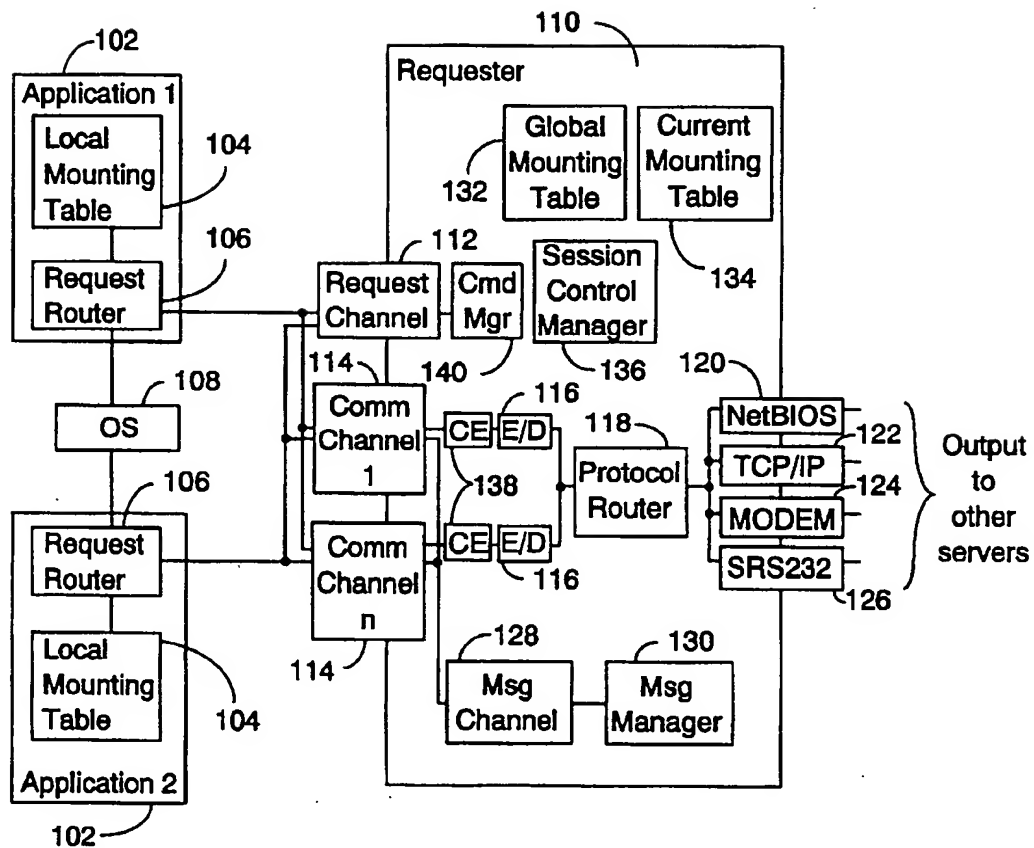
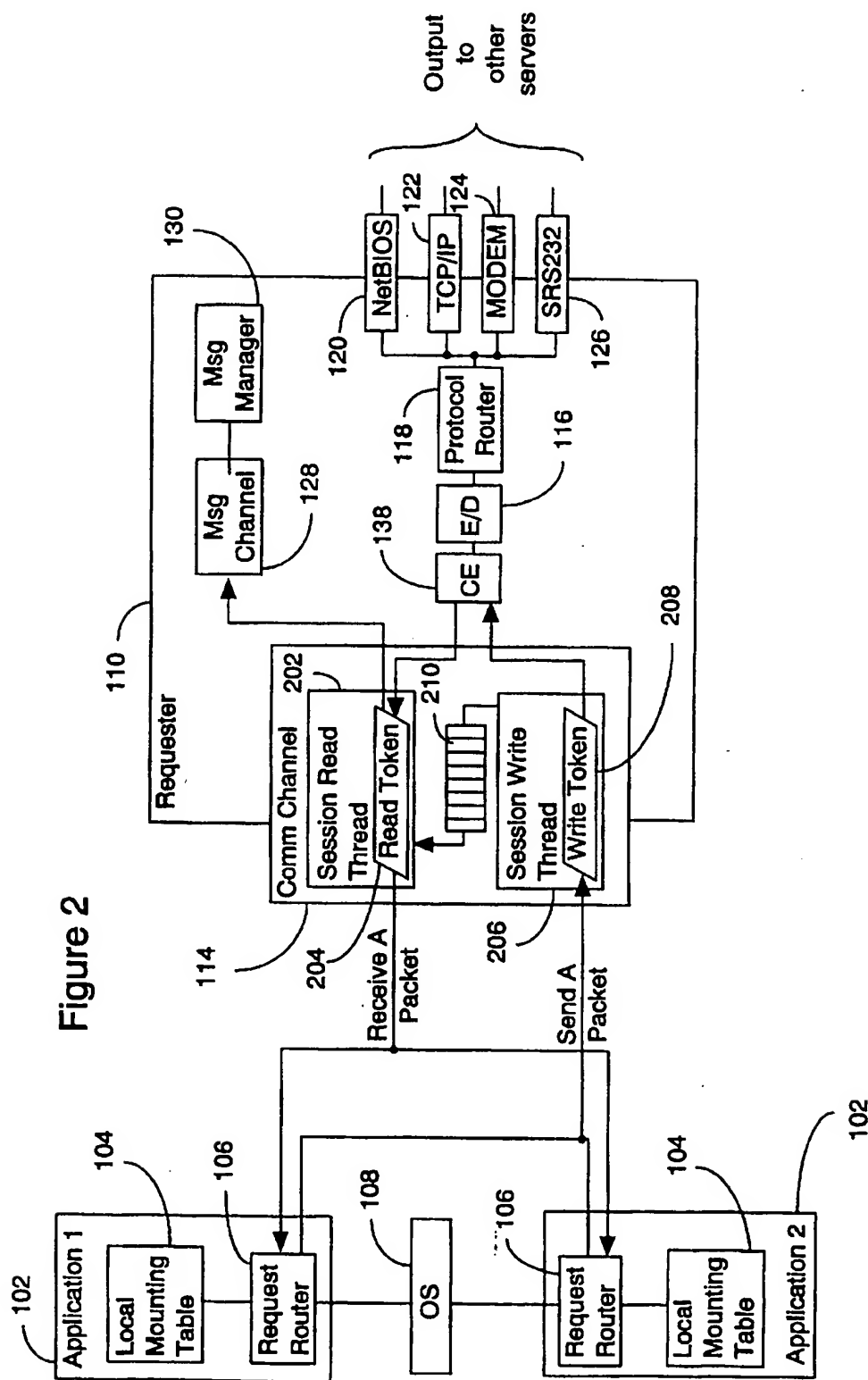


Figure 1



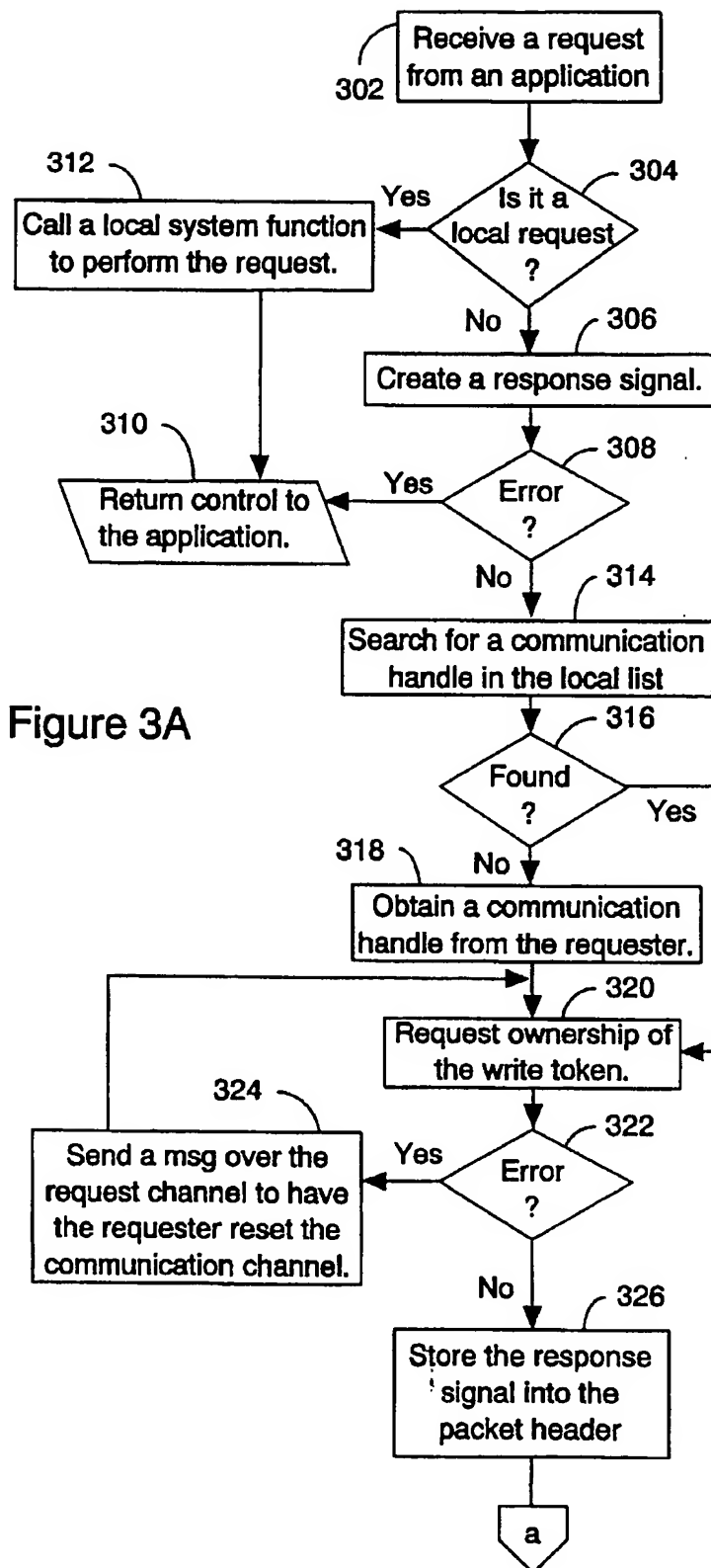
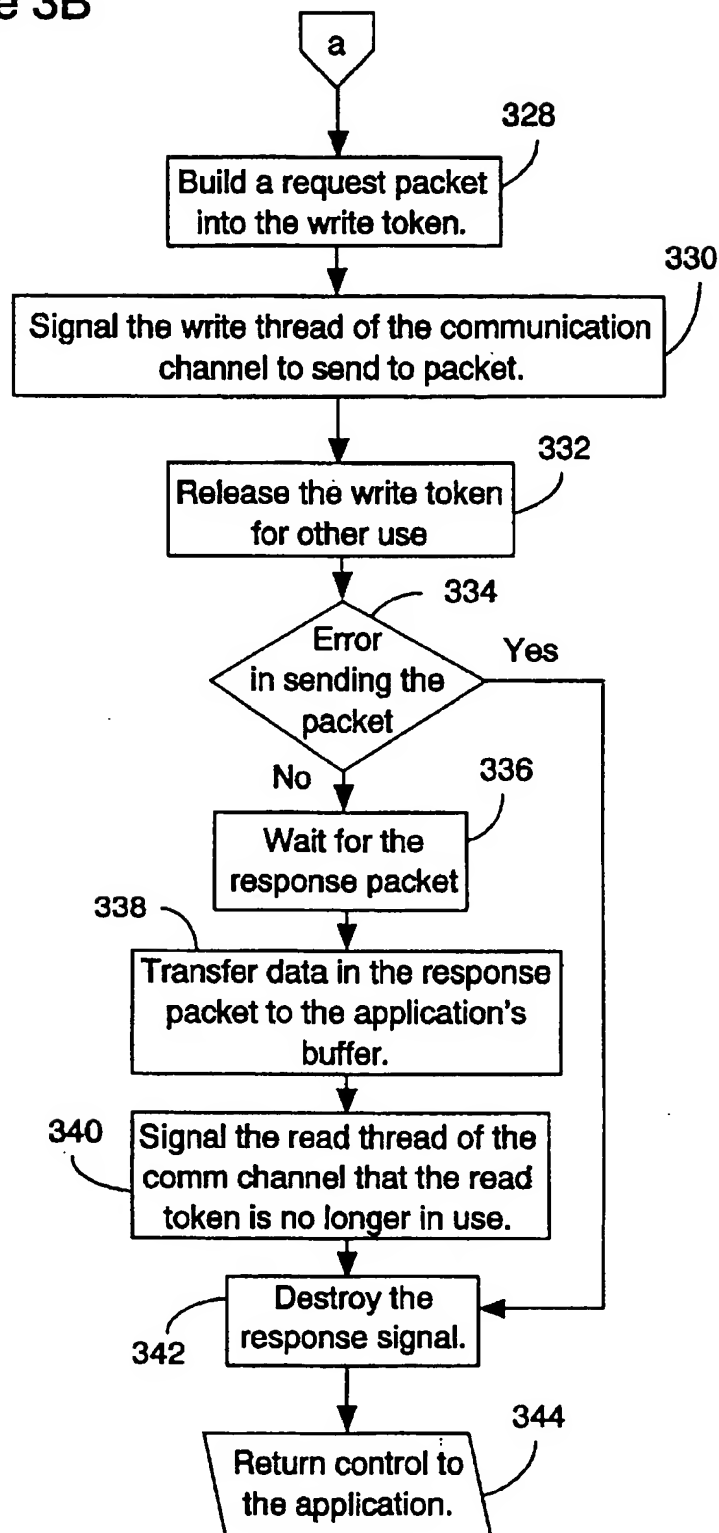




Figure 3B



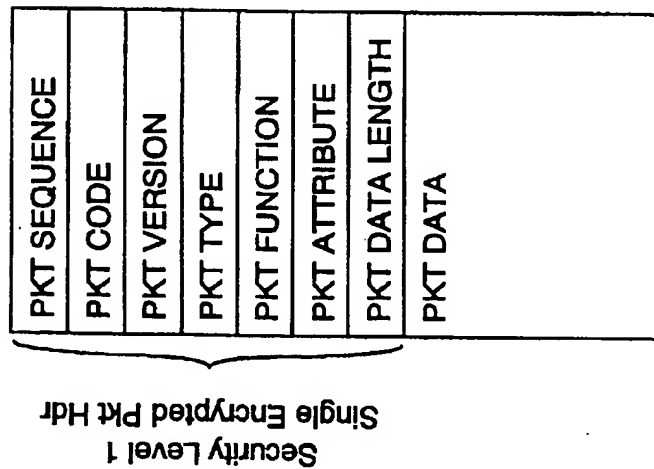


Figure 4A

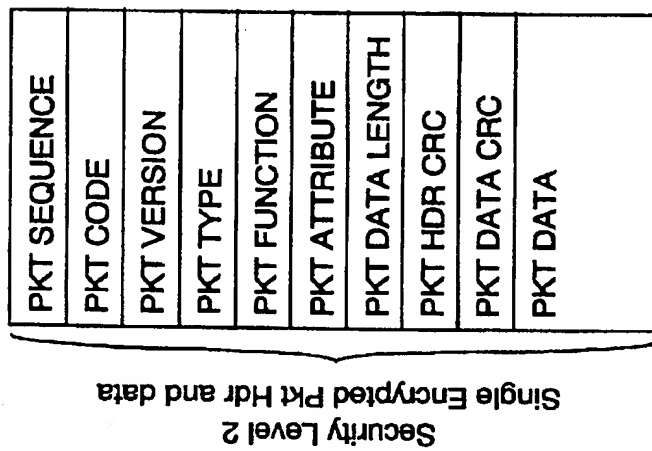


Figure 4B

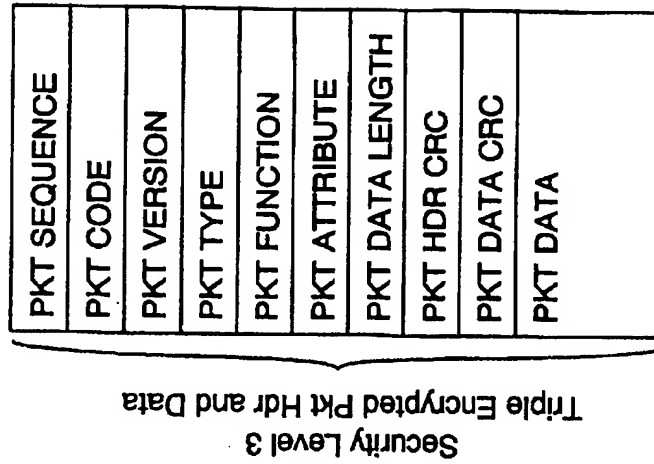


Figure 4C

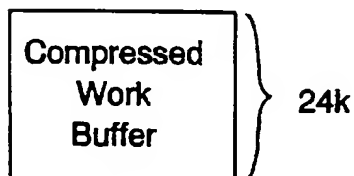
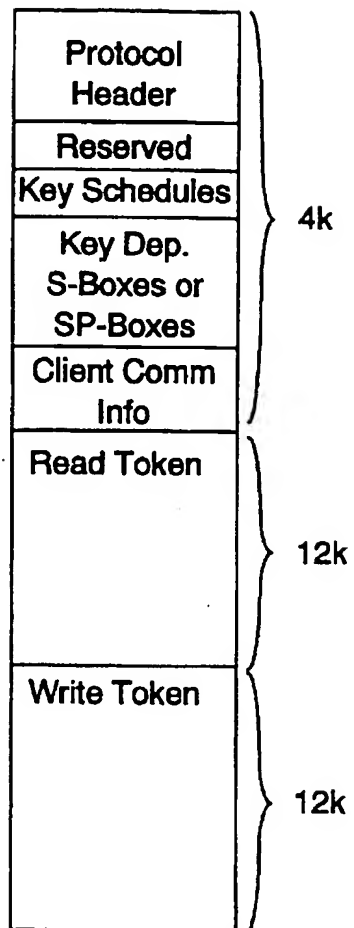


Figure 5A

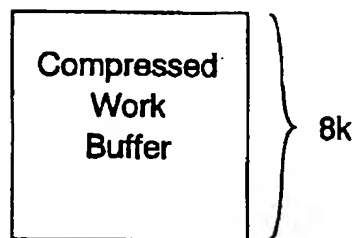
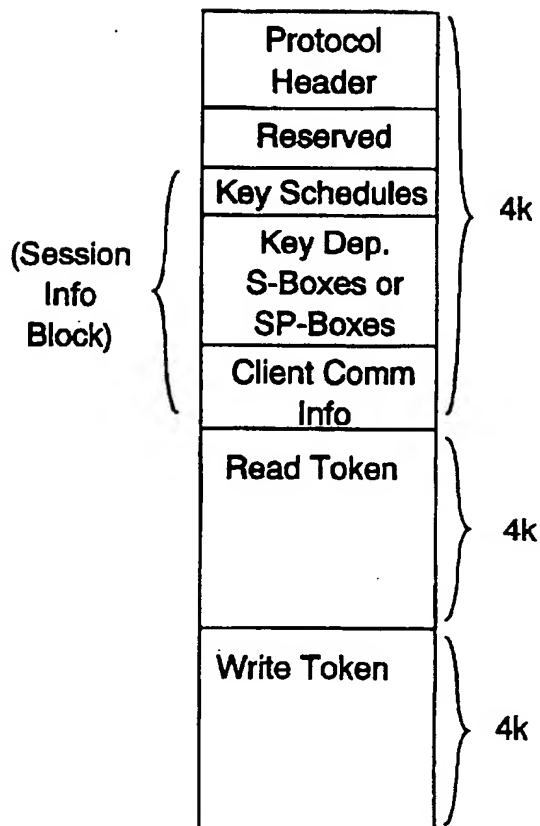


Figure 5B

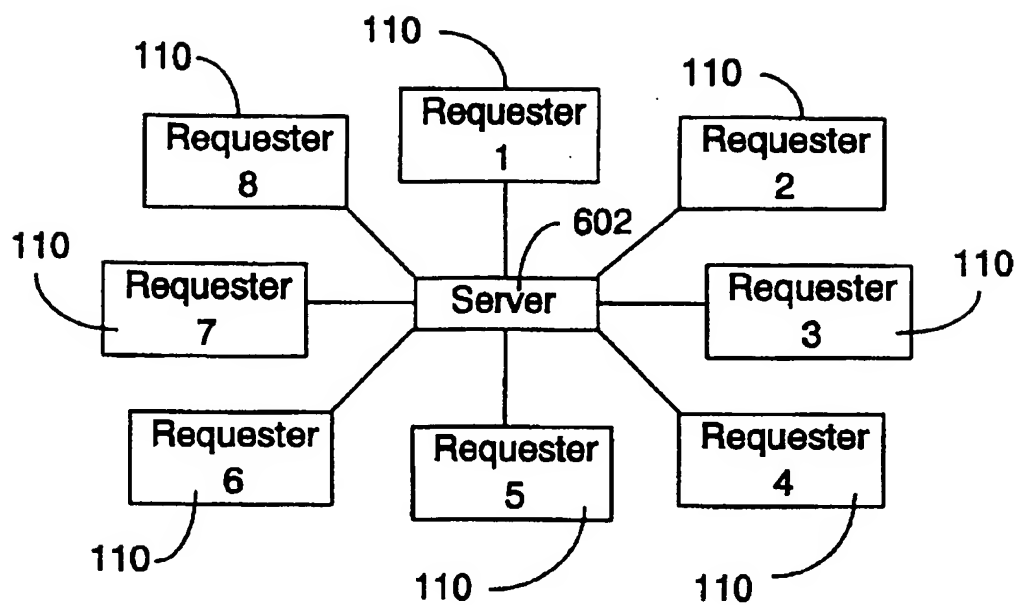


Figure 6

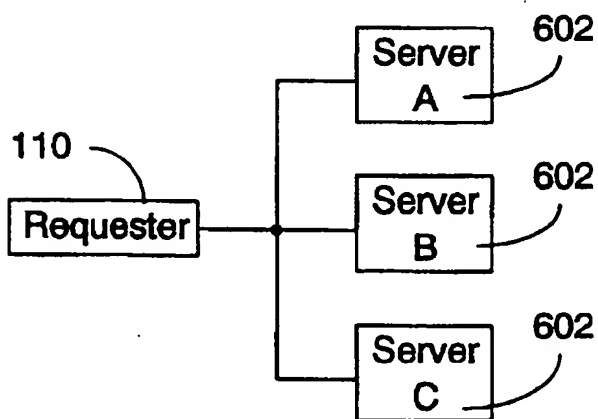


Figure 7

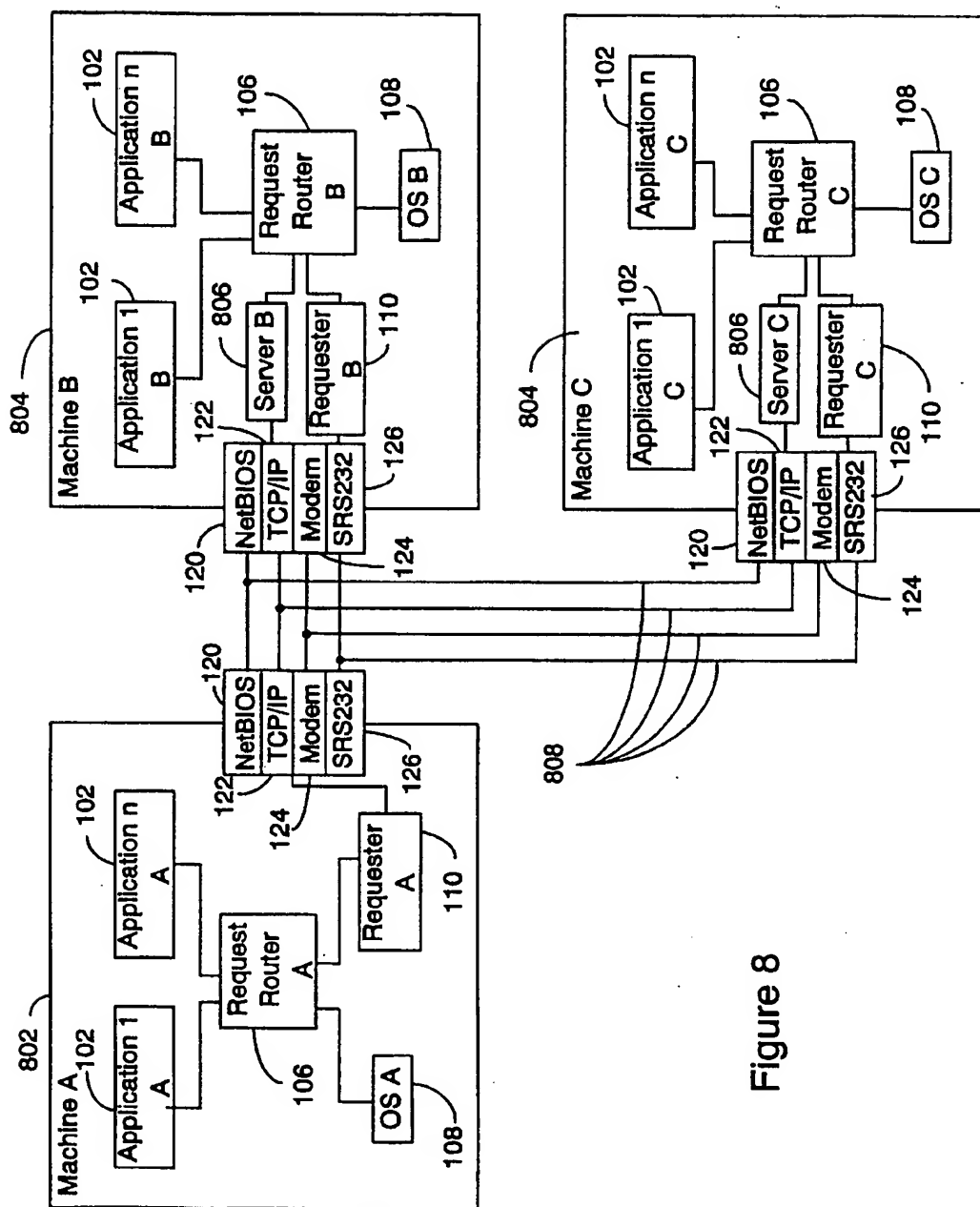


Figure 8

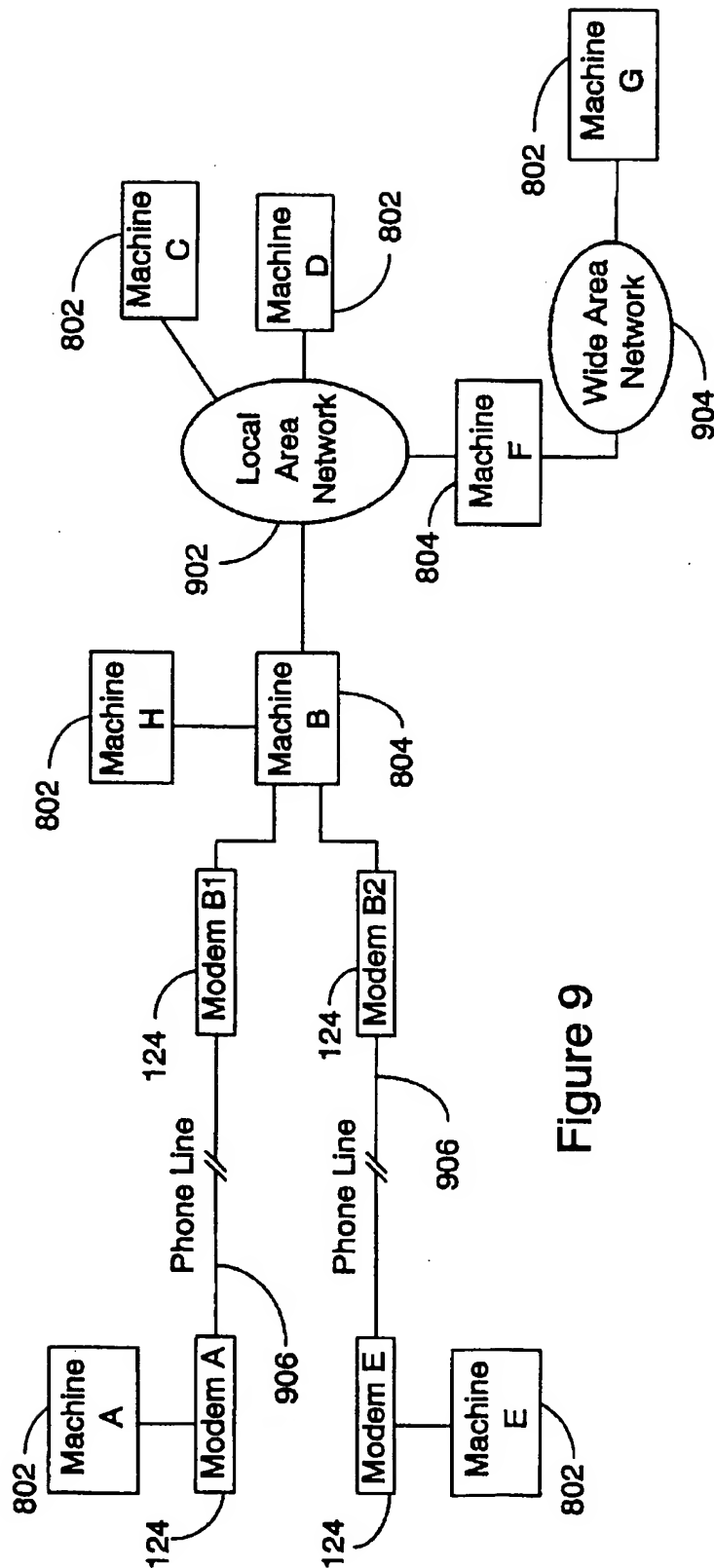


Figure 9

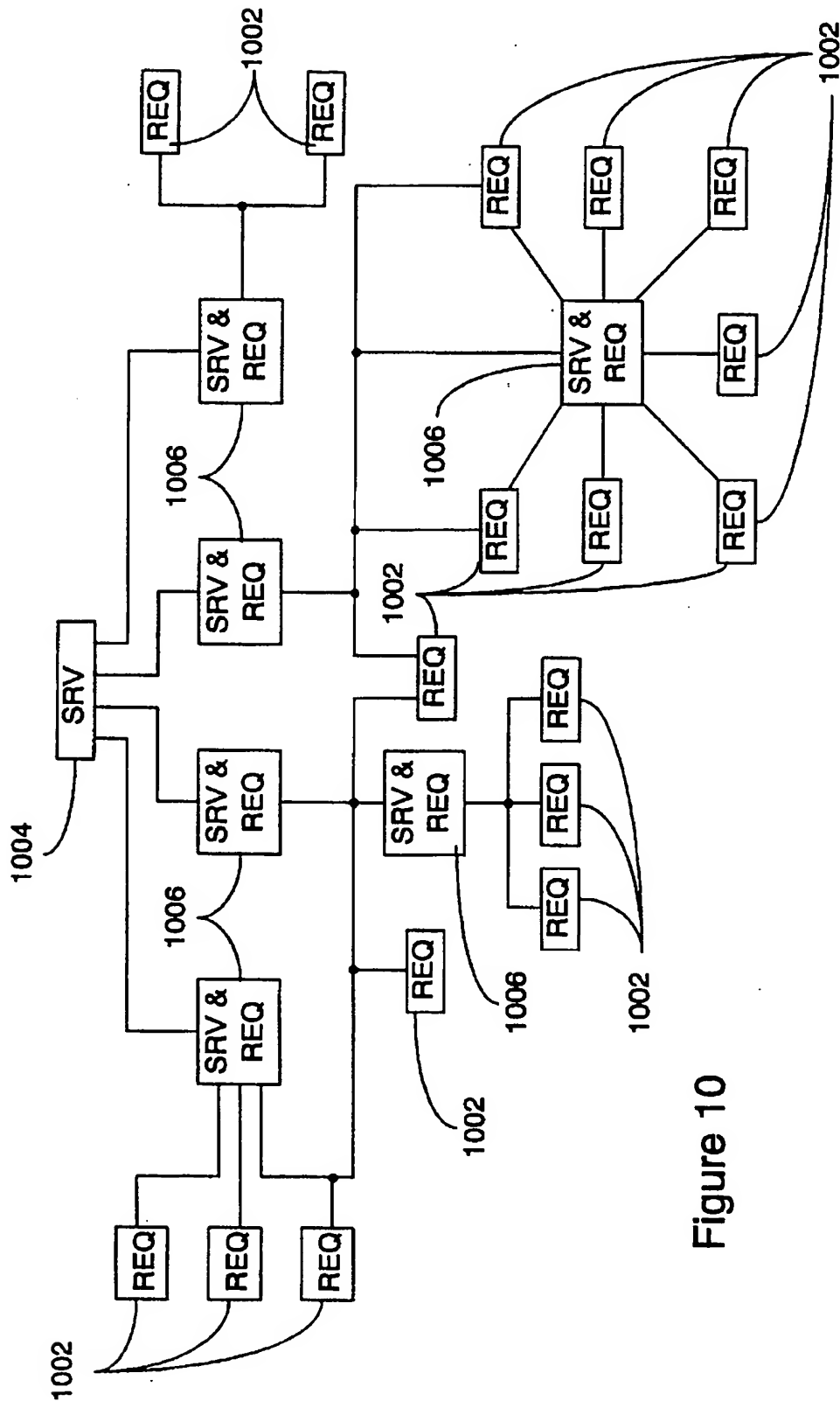
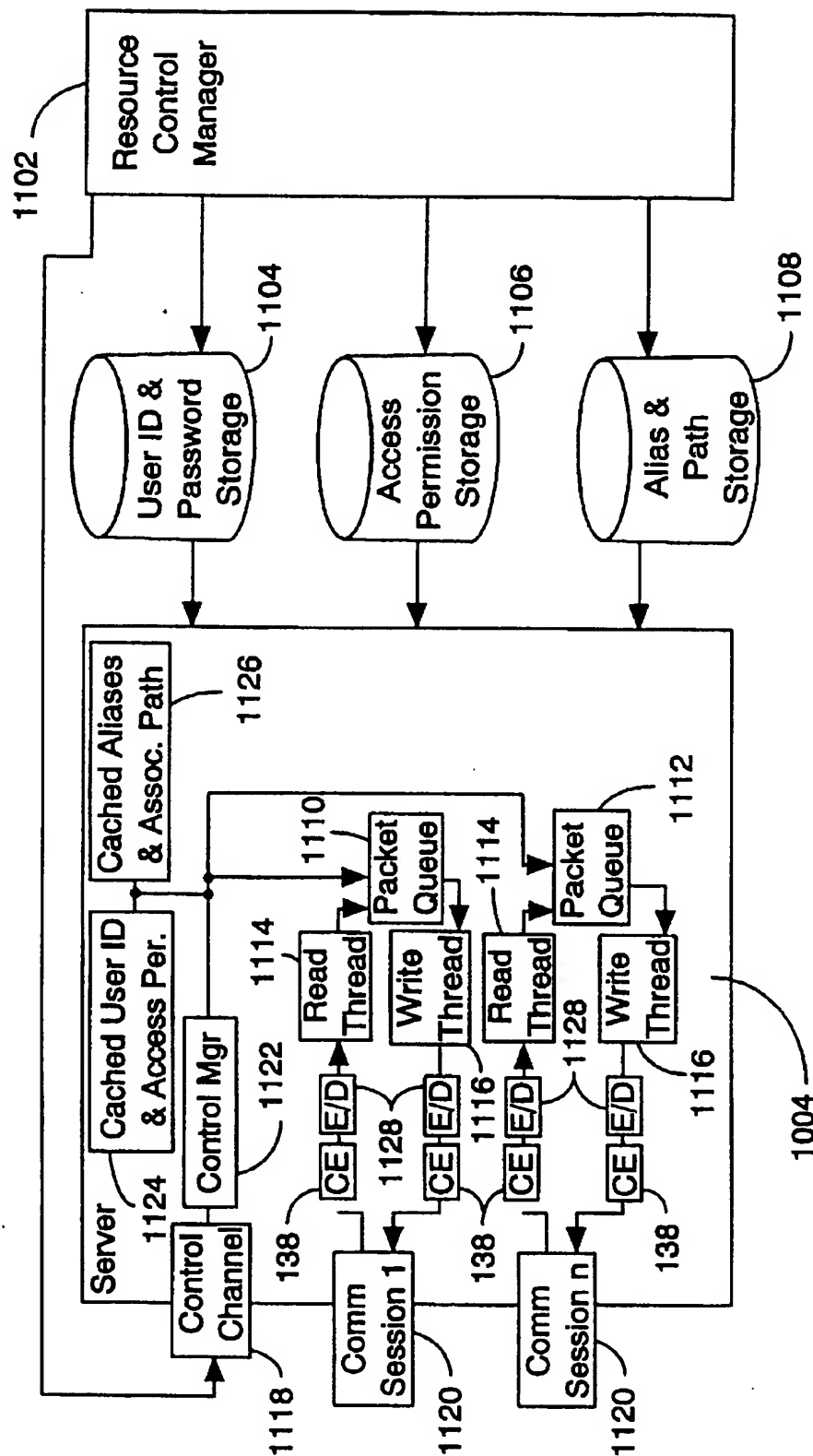


Figure 10



## Figure 11



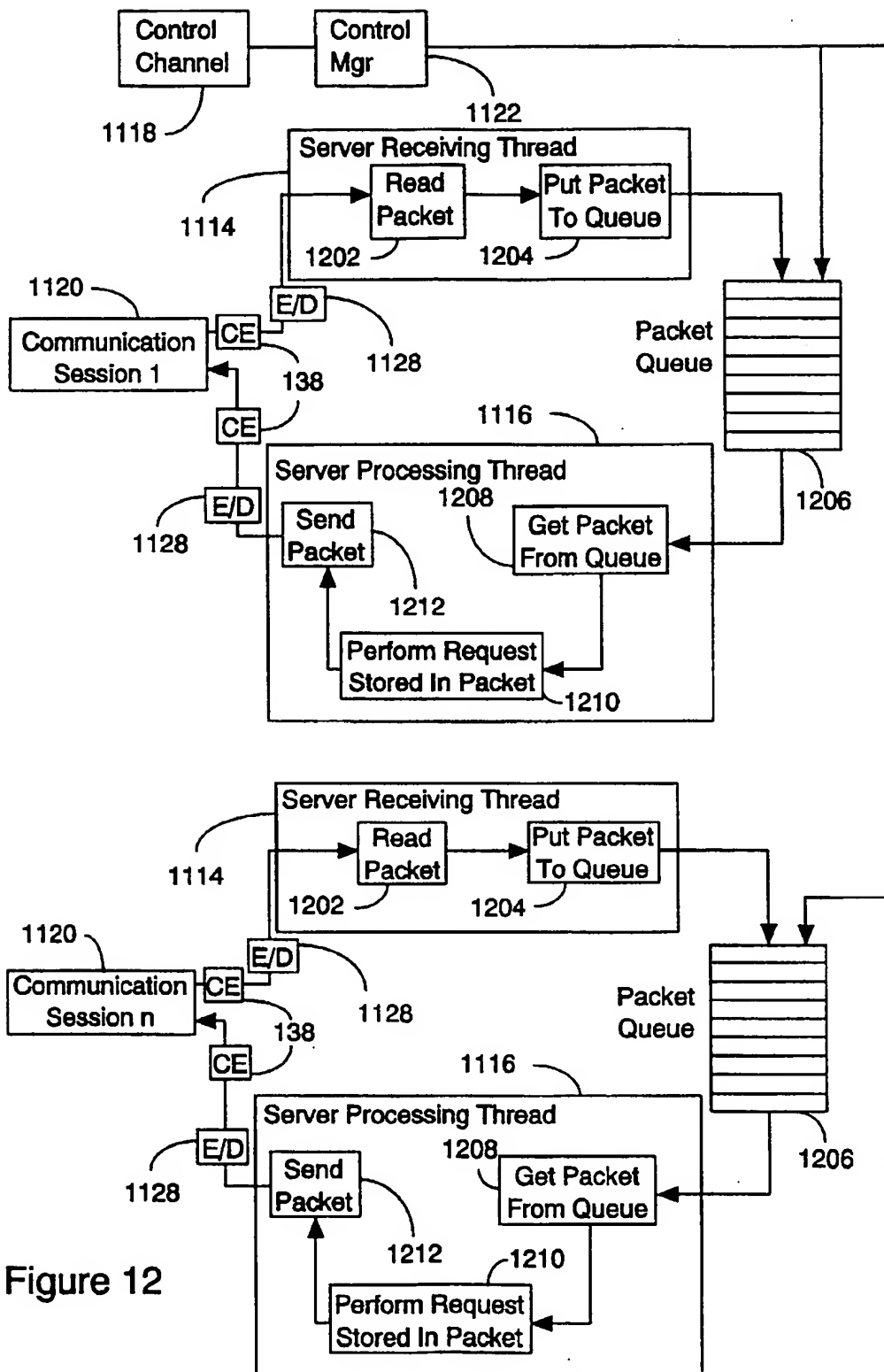


Figure 12

Figure 13A

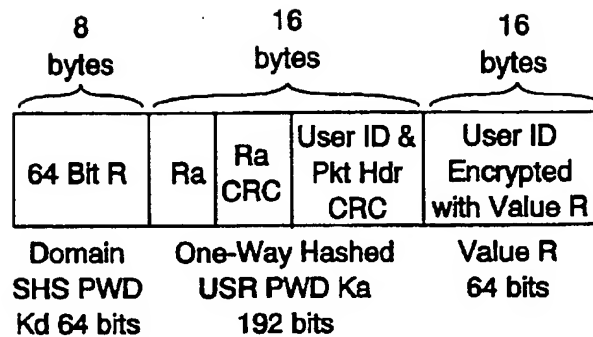


Figure 13B

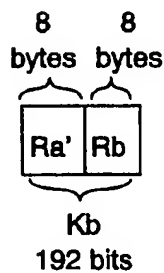


Figure 13C

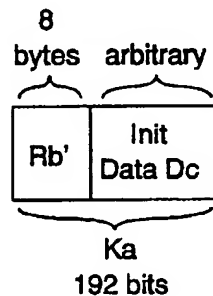


Figure 13D

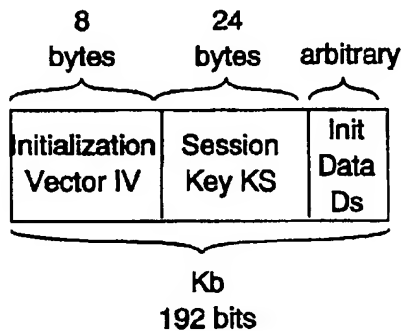
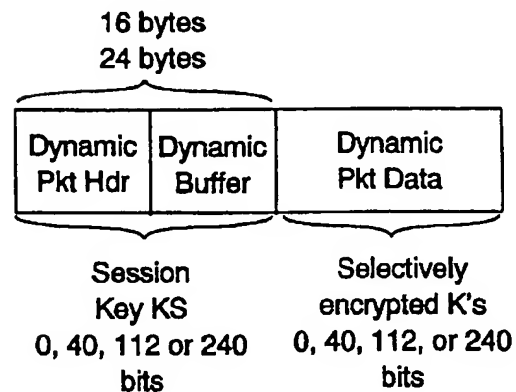


Figure 13E



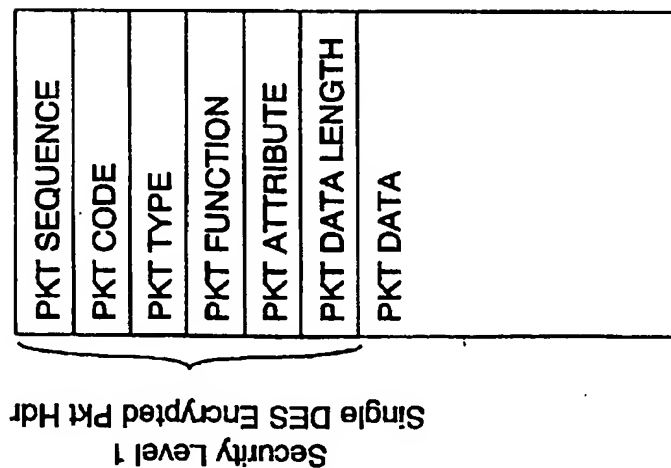


Figure 14A

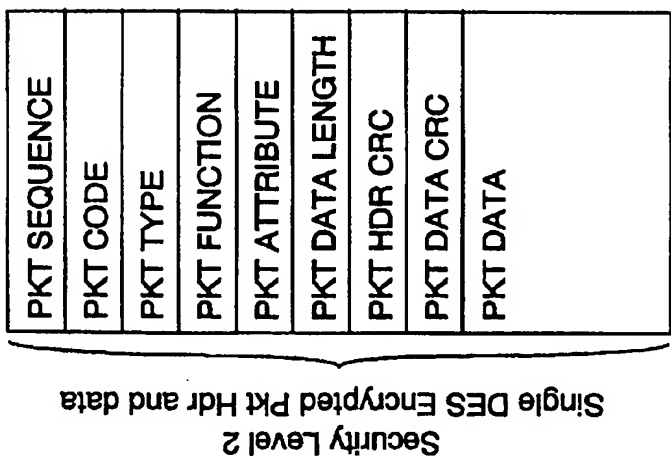


Figure 14B

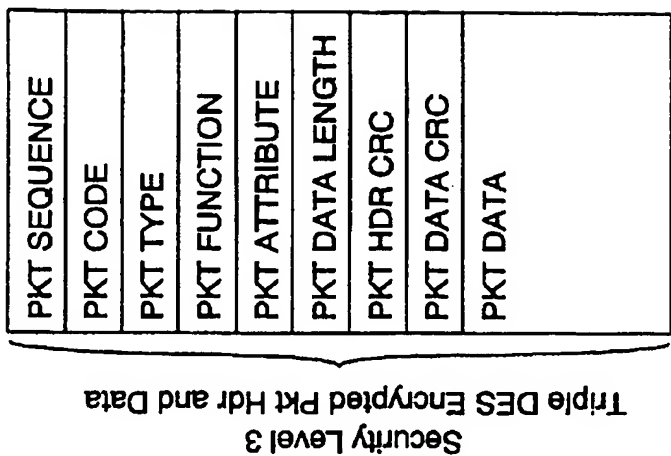


Figure 14C

Figure 15A

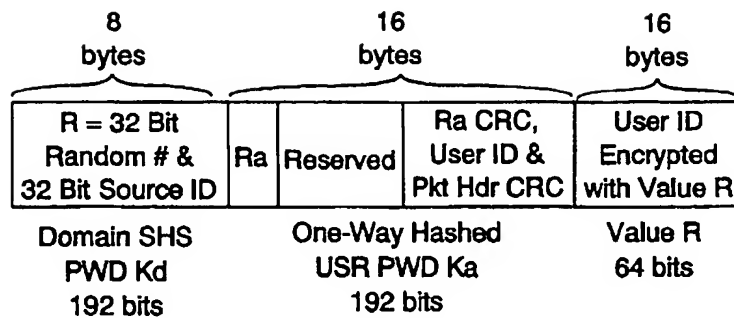


Figure 15B

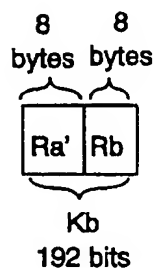


Figure 15C

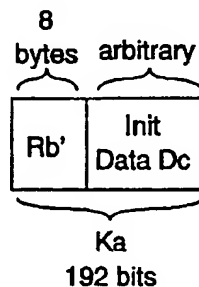


Figure 15D

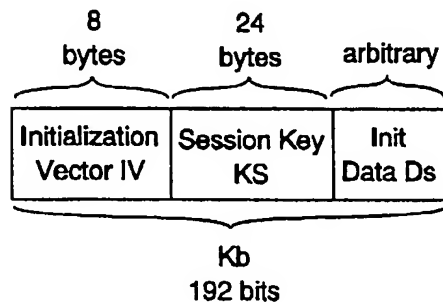
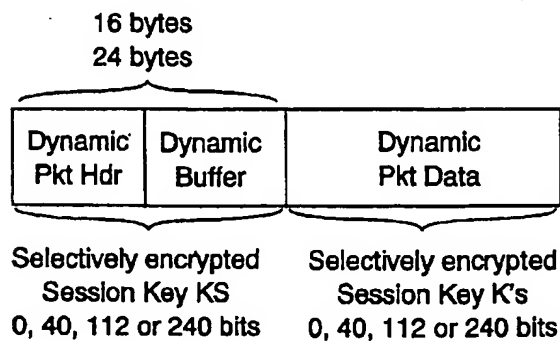


Figure 15E



# **NETWORK WITH SECURE COMMUNICATIONS SESSIONS** **CROSS-REFERENCE TO RELATED APPLICATIONS**

This application is related to and is a C.I.P. of the commonly owned copending application entitled Network with Secure Communications Sessions, filed Oct. 24, 1995, bearing U.S. Ser. No. 08/547,346 and naming Minhtam C. Nguyen, the named inventor herein, as sole inventor, the contents of which is specifically incorporated by reference herein in its entirety.

## **BACKGROUND OF THE INVENTION**

### **1. Technical Field**

The present invention relates to computer network security. In particular, it relates to networks which use dynamic packet headers and multiple levels of packet encryption to transfer data to and from a remote server or to and from another node in the local network.

### **2. Background Art**

The development of small independent systems such as personal computers has provided several benefits to users. By providing each user with their own processor and data storage, personal computers provide consistent performance and data security. A cost of these benefits is the inconvenience which results from the inability to easily access data by other members of an organization.

The use of mainframe systems, and the later development of alternative systems such as LANs (Local Area Networks) and servers reduces the inconvenience of making data available to all members of an organization, but results in unpredictable performance, and more importantly results in exposure of sensitive data to unauthorized parties. The transmission of data is commonly done via packet based systems which have user ID and password information in a header section. Interception of a packet with header information allows the interceptor to learn the user ID and password which will in turn allow future penetration of the user's system and unauthorized access to the user's data. It would be desirable to transmit user identification and password information in a manner which would be indecipherable to an unauthorized interceptor.

Data security is endangered not only by access by outside parties such as hackers, industrial spies, etc, but also to inadvertent disclosure of data to unauthorized members of the organization. For example, data exchange at certain levels of management may cause problems should the information be disclosed to the general employee population. Likewise, the transmission of personal information such as banking codes over networks has exposed individuals using online financial systems to the possibility of fraudulent access to their funds by third parties.

In addition to data security, the use of network systems such as LANs has created performance problems due to the queuing of requests from multiple locations and the unpredictable delays associated with queuing fluctuations. It would be advantageous if a system could provide not only data security, but also more consistent performance.

The prior art has failed to provide network systems which ensure that access to data is restricted to authorized parties while at the same time providing more consistent performance.

## **SUMMARY OF THE INVENTION**

The present invention solves the foregoing problems by providing a system which uses three way password

authentication, encrypting different portions of a logon packet with different keys based on the nature of the communications link. Nodes attached to a particular LAN can have one level of security for data transfer within the LAN while data transfers between LANs on a private network can have a second level of security and LANs connected via public networks can have a third level of security. The level of security can optionally be selected by the user. Data transfers between nodes of a network are kept in separate queues to reduce queue search times and enhance performance. Each communication session is assigned a key dependent S-boxes table for a high level of security.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

FIG. 1 is a diagram showing the connection between applications and the requester in a local system.

FIG. 2 is the diagram of FIG. 1 with a more detailed view of the requester.

FIGS. 3A-B are a flow diagram illustrating data transfer between the application and requester of the preferred embodiment.

FIGS. 4A-C are diagrams of the memory layout of packet headers used in the preferred embodiment.

FIGS. 5A-B are diagrams showing the memory layout of entries in the packet queue. FIG. 5A is the memory layout used for TCP/IP and NetBIOS. FIG. 5B is the memory layout used by modem or RS-232 communications systems.

FIG. 6 is a diagram of a multi-requester system with a single server.

FIG. 7 is a diagram illustrating a single requester attached to three servers.

FIG. 8 is a diagram showing a requester (machine A) interconnected with two servers (machines B-C).

FIG. 9 is a diagram illustrating multiple requesters connected to servers via local area networks (LANs) and wide area networks and public telephone networks.

FIG. 10 is a diagram illustrating multiple requesters connected to servers and server/requester systems.

FIG. 11 is a diagram illustrating the server used in the preferred embodiment.

FIG. 12 is a diagram illustrating the read/write threads and packet queues used by the server of FIG. 11.

FIG. 13A-D are diagrams illustrating the packet headers used in the logon procedure of the preferred embodiment.

FIG. 13E is a diagram illustrating the packet headers used during data transfer in the preferred embodiment.

FIGS. 14A-C are diagrams of an alternative preferred embodiment of the memory layout of the packet structures.

FIG. 15A-D are diagrams illustrating the logon packet structures used in the logon procedure of the alternative preferred embodiment.

FIG. 15E is a diagram illustrating the packet structure used during data transfer in the alternative preferred embodiment.

## **DESCRIPTION OF THE PREFERRED EMBODIMENT**

Prior to a detailed description of the figures, a general discussion of the operation of the preferred embodiment follows. A network can take a variety of forms. For example, it can be two personal computers communicating via modem; it can be a single LAN system within a particular facility; it can be a remote server or mainframe system with

communications links to individual terminals or personal computers; it can be a network of LANs or other servers each communicating with one another or through one another; or it can be any of the foregoing systems which use not only dedicated communications lines, but also non-dedicated communications (i.e. public networks such as the Internet) through a "firewall". The use of the term firewall herein refers to the requirement for increased levels of security to avoid the possibility of unauthorized data access by parties outside of the organization. Likewise, a machine in the network can act as a client or a server depending on the nature of the data transfer.

In the preferred embodiment, communication between a client and a server is as follows. The server waits for connection requests from clients on the network. The server can be started with one or more supported protocols to enable support of a variety of client types on the network. For example, the server protocols can include, among others, NetBIOS, TCP/IP, modem and RS-232. All of the foregoing protocols are well known in the art.

When a user on a client machine wishes to initiate a data transfer or other function, the client application activates a requester to access resources in the network. When the server receives a request from a client application, it activates a thread to process the request. A thread is an execution unit of an operating system. Operating systems used for this type of system are Microsoft Windows 95 (trademark of Microsoft Corporation), Microsoft Windows NT (trademark of Microsoft Corporation), IBM OS/2 (trademark of IBM Corporation). These systems may use multiple session protocols such as NetBIOS and TCP/IP or single session protocols such as modem or RS-232.

In single session protocols such as modem and RS-232, the same thread is used to process the request from a client since a serial port can act a server or client, but cannot simultaneously act as a server or client. Multiple session protocols create a new thread, referred to as an original thread, and wait for a request from a client. When a request is received, the thread is referred to as a server processing thread which is used to process the client logon.

After the logon is successfully completed, the server processing thread creates a packet queue and a packet thread to receive incoming packets and place them in the packet queue. The server then waits for packets to arrive. On the client side, the client creates a session write thread to initiate contact with the server. In addition, the client creates a second thread which is referred to as the session read thread. This thread is used to receive packets sent from the server to the client.

To use resources on the network, users must first logon the server to prove their identity. A logon request is sent from the client's logon application to the requester on the client computer. Before logon data can be exchanged between the applications and the requester, a command manager is created by the requester to accept application requests. The command manager is responsible for housekeeping requests within the client computer.

In the preferred embodiment, the logon procedure uses a three way authentication to prevent the password from being transferred over the computer and also to allow both the client and the server to authenticate each other. In addition, the authentication procedure prevents unauthorized penetration of the system security by detecting the replaying of packets by third parties.

The three way authentication system encrypts the very first logon packet with different keys for each part of the packet as follows.

The first step takes place at the client computer as follows.

- 1- The client generates a 32 bit random number value which is concatenated to a predefined 32 bit constant to form a 64 bit value R.
- 2- The CRC signature C1 of the 64 bit value R and the user ID is calculated. This signature value allows detection of packet manipulation.
- 3- The 64 bit value R is used as a DES key to encrypt the user ID. This makes the user ID look random for each logon packet.
- 4- The client generates a 192 bit key K from the server name to encrypt the 64 bit value R.
- 5- The client generates a key Ka from the user ID and password using a one way hash function such as the Secure Hash Standard (SHS) specified in the Federal Information Processing Standards Publication 180 (FIPS PUB 180).
- 6- The client generates a random number Ra, calculates its CRC signature C2, and encrypts them with the signature C1 using the key Ka. This signature is used to validate the key Ka by the server.

The second step in the process takes place at the server. When the server receives the first logon packet it decrypts the packet as follows.

- 1- The server generates a key K2 from its machine name and the SHS to decrypt the packet header for identification. If the packet header does not contain the predefined constant, the user is unauthorized. This occurs when an unauthorized user tries to access the server over the phone line but does not know the server name (since the phone number is a public record but the server name is private).
- 2- If the user is authorized, the server uses the decrypted 64 bit value R in the packet header as a key to decrypt the user ID.
- 3- The server then uses the user ID to search a database for an access record. If the access record cannot be found, the user has entered an invalid ID and the session is terminated. If the access record is found, the server verifies if the user is allowed access to network resources at this date and time.
- 4- If access date and time are verified, the server retrieves an associated one way hashed password Kb from an encrypted password file to decrypt the random number Ra and the CRC signatures. The password file is encrypted with a key Kk which is selected by the system administrator at installation.
- 5- The random numbers Ra and the CRC signatures are then decrypted. The server calculates the CRC signature of the packet header, the user ID and the random number Ra. If the calculated signatures match the decrypted signatures C1 and C2 stored in the packet, and if password Ka matches Kb, the server manipulates the client random number Ra with a predefined formula, generates a random number Rb, and encrypts both random numbers Ra and Rb with the password Kb before sending the first logon response packet to the client.

The third step in the process takes place at the client computer as follows.

- 1- The client decrypts the first logon response packet.
- 2- The client manipulates the random number Ra with the predefined formula and compares it with the one returned from the server. If the numbers match, the client knows that it is connected to the correct server, not a fraud server from which an eavesdropper has captured transmissions from the previous logon and is echoing packets back to the client computer.
- 3- The client manipulates random number Rb with another predefined formula and concatenates it with the client's

initiating data (i.e., the client initial packet sequence number, the encryption and compression mode for the session, and the operating system platform ID) to form a second logon packet. The operating system platform ID is useful for selecting protocols and data formats when a particular client or server is communicating with systems that may have any one of a variety of operating system software programs running. The client would typically request encryption and compression mode for the session. However, the server may indicate that the particular modes requested are not available.

- 4- The client then encrypts the second logon packet and sends it to the server.

The fourth step in the process takes place at the server computer as follows.

- 1- The server decrypts the second logon packet.
- 2- The server manipulates the random number Rb with the same predefined formula used by the client and verifies if the random numbers are matched. If the random numbers match, then the server knows it is communicating with an authorized client and that the first logon packet was not a replayed packet.
- 3- The server saves the client initiating data, generates a session key Ks and an initialization vector IV. In the preferred embodiment, Ks and IV are generated using a formula similar but more secure than the one specified in Appendix C of the ANSI X9.17 standard.
- 4- Ks and IV are sent to the client along with the server initiating data (i.e., the server initial packet sequence number, supported and/or approved encryption and compression modes for the session, and the server operating system platform ID).

The client and server initial packet sequence numbers are used to detect packet deletion and insertion for data exchanged after the logon procedure.

The fifth step in the process takes place at the client computer as follows.

- 1- The second logon response packet is decrypted by the client.
- 2- The client encrypts Ks and IV with its own key and saves them in memory for future communication with the server. The logon procedure completes here.

After the logon procedure is successfully completed, all packet headers are encrypted using the session key Ks and the IV. The packet headers are encrypted to prevent intruders from deleting, inserting, modifying, and/or replaying the packets which may have been captured while data was exchanged over communication lines.

For ease of illustration, the following symbols can be used to illustrate the logon process:

Where:

C=a client

S=a server

E=a symmetric cryptosystem such as DES

K=an encryption key generated from the server name  
R=a 32 bit random number concatenated with a pre-defined constant

Ka=a 192 bit key one way hashed from the user ID and password

Ra=a 64 bit random value generated by C

f( )=a hash function such as CRC to calculate the signature

g( )=a hash function such as CRC to calculate the signatures

UID=user IDs

Kb=a 192 bit one way hashed key retrieved from a database

ha( )=a hash function to manipulate the random number Ra

Rb=a 64 bit random value generated by S

hb( )=a hash function to manipulate the random number Rb

Dc=client initial data

IV=an initial chaining vector for encryption

Ks=a session encryption key

Ds=server initial data

R'a=ha(Ra)

R'b=hb(Rb)

The logon procedure may be listed as:

1. C to S: EK(R)+EKa(Ra,f(Ra,g(R,UID)))+ER(UID)

2. S to C: EKb(R'a,Rb)

3. C to S: EKa(R'b,Dc)

4. S to C: EKb(IV,Ks,Ds)

An important advantage of the authentication procedure used by the preferred embodiment is that both the client and the server verify each other as legitimate without sending the password. In addition, the use of a second set of logon packets which contain different encrypted random numbers precludes access by an unauthorized intruder who merely replays intercepted packets.

The heart of this authentication procedure is in the middle part of the logon packet, which contains the random number Ra and the CRC signatures. Since the CRC signature of the random number Ra is encrypted and sent along with the logon packet, the server can authenticate the user right on the first logon packet. The manipulation of the random numbers Ra and Rb in the challenge-response fashion is to help the server defeat the replaying of the logon packet and to allow the client to authenticate the server and to defeat packet replaying as well.

The 32-bit random number in the packet header is used to make the packet header and the user ID look different for every logon packet. The one-way hashed server name is used as a key to quickly detect invalid logon packets before searching the database. This case may occur frequently when the modem protocol is activated to wait for data transferred over a telephone line (i.e., a wrong number is dialed by accident or a call generated by a manual or automated telemarketing company is being received).

In addition, the server name is isolated from the user ID and password when creating a one-way hashed password to allow the portability of the database. For example, when a business grows, another server may be needed at another location and the database can be easily transferred to the new server. Of course, it would be less time-consuming to delete unauthorized users from the database than to add authorized users to the new one. To better protect the valuable information in the database, a password is required before access to the database is granted. More important, the database can be shared among servers. For example, a server Sb can receive the first logon packet and forward the user ID to a database server Sc within a private network for verification. If an access record is found and the user can access the server Sb at this date and time, the database server Sc returns the encrypted one-way hashed password Kb to the server Sb. The server Sb then continues the challenge-response as if the password Kb is returned from a local database. Note that the database server Sc can encrypt the one-way hashed password Kb with the session key defined for communication between the server Sb and Sc before sending it across the private network is security is desired.

In comparison to prior art systems, the design of this invention provides the server a better opportunity to resynchronize itself if the first logon packet is invalid since the

receiver of the authenticating packet is in control of what is next, not the sender. On the other hand, in the prior art the sender is in control of what is next. For example, the sender generates a public key, encrypts it with a shared secret key and sends it to the receiver. If the secret key is invalid, the receiver cannot detect it. Thus, a certain number of packets must be received before the receiver can resynchronize or the receiver might have to use a timeout to resynchronize itself.

Finally, the logon protocol of the preferred embodiment is more suitable for a client/server distributed environment, because this logon protocol allows both client and server to authenticate each other without sending the user password across the communication media and prevent intruders from deleting, inserting, modifying, or replaying the logon packets. In addition, if the logon procedure fails at any point, the server releases all resources and destroys the connection without sending the response packet at that point, i.e., if the user enters a wrong server name in the very first logon packet, nothing is sent out from the server to prevent the user, a potential intruder, from knowing anything about the server. Note that this mutual authentication technique requires the client machine to have a local CPU so that the password will not be transmitted over the network before being encrypted.

The client can now perform a mounting procedure to link a network resource on the server to a virtual disk or it can identify a network resource with the following format \\servername\netname. The communication protocol selected at logon is used for communication between a particular client and the server. This method allows communication between a client and network domains, between a network domain and other network domains using multiple communication protocols. Therefore multiple clients can communicate with a single server, each client using a different protocol if desired. Also a single client can communicate with multiple servers, also using different protocols for each server.

Referring to FIGS. 1 and 2, these figures illustrate the interconnection between a client and a server. FIG. 2 is a more detailed view of the system of FIG. 1.

To perform a file transfer operation, an application 102 calls a request router 106. The request router first verifies if the application 102 requests a local or remote resource. This verification is performed using a local mounting table 104 which the request router 106 obtains from the requester 110 when the application 102 is first started.

If the resource is local, the request router 106 calls a local system function call to perform the request and returns the control to the application 102. However, if the resource is remote, the request router 106 first searches its local list to see if the needed communication handle is already stored in the list. This communication handle contains information of the read 204 and write 208 tokens (shown in FIG. 2) and their associated resources. If the communication handle is not found in the local list, the request router 106 sends a message to the requester 110 over the request channel 112 to obtain the handle. Once the handle is obtained, the request router 106 creates a response signal, i.e., a return address, requests the ownership of the write token 208, stores the response signal into the packet header, builds a packet based on the application's 102 request into the write token 208, and signals the session write thread 206 of the communication channel 114 that there is a packet to send.

If the application data is larger than the packet capacity, the request router 106 can send multiple packets in a series at this point. After the packet is sent to the server, the request

router releases the write token for use by another thread in the same process or a different process. If the packet was sent to the server successfully, the request router 106 waits for the corresponding response packets, i.e., a packet can cause multiple response packets returned from the server.

When a response packet arrives, the session read thread uses the response signal to tell the corresponding request router that its response packet has come and is available in the read token. At that time, the read token is accessed exclusively by the designated request router. The router then transfers data in the response packet directly to the application's buffers and signals the session read thread 202 of the communication channel 114 that the read token 206 is no longer in use so that the session read thread 202 can re-use the read token 206 for other incoming packets. Finally, after all response packets of a request packet have arrived, the request router 106 destroys the response signal and returns control to the application 102. The final response packet is determined by a bit in the packet attribute.

The request router 106 sends a message to the command manager of the requester 110 to request the communication handle containing information of the read 204 and write 208 tokens and their associated resources. If the handle already exists, it is passed to the request router 106 immediately after the requester 110 increments the access count of the handle. However, if the handle does not exist at that time, the requester 110 will load the appropriate communication library, allocate the tokens 204, 208 and their associated resources, create a communication channel consisting of a session write thread 206 to perform auto-logon, create a session read thread 204 for the communication channel 114 if auto-logon is successful, and increment the access count of the handle before passing it to the request router 106.

After receiving the handle, the request router 106 saves the handle for use during the entire lifetime of the application. When the application 102 terminates, the request router 106 will signal the requester 110 of the event so that it can decrement the access count of the handle. When the access count is zero for a certain period of time, the session manager of the requester 110 will drop the communication session, release the tokens 204, 208 and their associated resources, and unload the communication library. Thus, this method allows resources to be allocated upon demand and released when no longer in use. Furthermore, the request router 106 can translate and format data in the application timeslices while the requester 110 is communicating with communication devices 120, 122, 124, 126 to better use the CPU time.

The request router 106 can also perform any preparation necessary to transfer the application 102 request to the requester 110 before requesting the ownership of the write token 208 to reduce the time it takes to access the write token 208. In addition, the request router 106 remembers resources for one application 102 at a time. Thus, it reduces the time to search for the needed information. With this method of sending and receiving packets, data can be exchanged asynchronously between a client and a server with minimum resources in a minimum time. In addition, request packets can be accumulated on the server for processing while the previous response packet is processed by the communication devices 120, 122, 124, 126 or traveling over the network.

Message channel 128 and message manager 130 are used to control system messages transmitted in the system. Current mounting table 134 and global mounting table 132 are used to identify usage of system resources. The session control manager is used to control each session between a client and a server.



In order for the Requester 110 to perform a secure automatic logon after the Session Control Manager 136 has dropped a session due to being idle, the Requester 110 must save user ID and password in the Global Mounting Table 132. However, not the original password is stored in the table; but the one-way-hashed key Ka generated from the user ID and password is saved. This key Ka is encrypted with another key generated randomly every time the Requester 110 is started to further protect it. The original password is erased from memory immediately after it is used to generate the key Ka.

When an application need to communication with the remote server, the request router 106 will download the Current Mounting Table 134 into the Local Mounting Table 104 in the application's process space by sending a download request to the Command Manager 140 of the Requester 110. The name of the remote server can now be retrieved from the Local Mounting Table 104 by the Request Router 106. This remote server name is used to request a communication handle from the Requester 110.

When the Requester 110 performs automatic logon due to a request from a Request Router 106 for a communication handle, the Command Manager 140 searches the Global Mounting Table 132 to find an appropriate server name. If the server name cannot be found, the user has not performed logon manually. However, if the server name is found, the Command Manager 140 will create a Session Write Thread 206 and its associated resources, decrypt the encrypted key Ka and store logon information such as the user ID, the key Ka, and the server name into the Session Info Block as shown in FIG. 5A-B. The auto-logon procedure is then performed by the Session Write Thread 206. After the logon procedure is successfully completed, the Session Write Thread 206 will automatically create the Session Read Thread 204 as described earlier. A return code is then returned to the Command Manager 140 of the Requester to indicate success or failure. If the auto-logon is successful, the Requester 110 returns the communication handle to the Request Router 106. Otherwise, an error code is returned to the Request Router 106. Compression engines (CE) 138 are used to compress data for performance reasons.

If the user has selected compression for the communication session, a compression work buffer will be allocated by both the client and server during the logon procedure. Therefore, minimum resources are allocated for better performance, because the compression work buffer is only allocated when it will be used.

FIG. 3A and B is a flowchart which illustrates the transfer of information in a session after the logon procedure has completed. When a resource request 302 is made, the system 304 first tests to see if it is for a local resource 304. If so, a local function is called 312 and control is returned 310 to the application. If it is not a local resource, the system creates a response signal 306. If the response signal 306 cannot be created, control is returned to the application. If it is, then the local list is searched 314 for the communication handle. If the communication handle is not found 316, a communication handle is obtained 318 from the requester and then ownership if the write token is requested 320. However, if the communication handle is found 316, then ownership if the write token is immediately requested 320.

If no error occurs when the request for ownership of the write token is made 322, then the response signal is stored in the packet header 326, a request packet is built into the write token 328, the write thread sends the packet, and the write token is released 332. If an error is detected when the packet is sent, the response signal is destroyed 342 and

control is returned 344 to the application. If no errors occur during packet transmission 344, then the system waits 336 for the response packet, the data in the response packet is transferred 338 into the application's buffer, the read token is released 340, the response signal is destroyed 342 and control is returned 344 to the application.

FIGS. 4A-C illustrate the memory layout of the packets used in the preferred embodiment. FIG. 4A illustrates a packet as encrypted by security level 1. In security level 1, the packet header is encrypted using single s<sup>3</sup>DES encoding. This level of security incurs the least amount of overhead and is preferably used in more secure environments such as LANs. However, if the remote client or server is outside the U.S., then the standard DES is used with a 40 bit key to weaken security to comply with U.S. law.

FIG. 4B illustrates a packet as encrypted by security level 2. In security level 2, the packet header and data are encrypted using single s<sup>3</sup>DES encoding. This level of security incurs slightly increased overhead as compared to security level 1, but provides an increased level of security for less secure environments such as wide area networks.

FIG. 4C illustrates a packet as encrypted by security level 3. In security level 3, the packet header and the data are encrypted using triple s<sup>3</sup>DES encoding. This level of security incurs the most overhead as compared to security levels 1 and 2, but provides the highest level of security for insecure environments such as public telephone networks or the Internet.

To protect data exchanged over communication sessions, the preferred embodiment provides two different encryption schemes available to the user at logon. The first scheme is the single s<sup>3</sup>DES and the second scheme is the triple s<sup>3</sup>DES similar to the one specified in the ANSI X9.17 and ISO 8732 standards but with 192-bit keys. In addition, the preferred embodiment applies the standard Cipher Block Chaining mode specified in the FIPS PUB 81 to better protect the data. Once an encryption scheme is selected, data exchanged over all sessions connected to a network domain are encrypted regardless of the communication protocols being used by the sessions. The price to paid for the encryption is minimum anyway since the preferred embodiment encrypts 500,000 bytes per second when running on a Pentium 66MHz processor. The operating system used can be any suitable personal computer operating system such as a Microsoft (TM) Windows 95 (TM), IBM (TM) OS/2 Warp (TM), Unix, etc. If the server is a large system, any one of a number of suitable mainframe operating system software may be used.

In addition to the above encryption schemes, the preferred embodiment employs a dynamic packet header technique to provide extra securities based on the security level selected by the user at logon. If a security level 2 is selected, the packet header and data are encrypted with s<sup>3</sup>DES and the packet header is changed to 24 bytes to carry the CRC signatures of the packet header and data for authentication. However, if a security level 3 is selected, the packet header and data are encrypted with s<sup>3</sup>DES using three different keys. Finally, if security level 1 is selected, the packet header remains at 16 bytes and no signature is verified for a better performance but the packet header is encrypted with s<sup>3</sup>DES to provide security against other threats. Thus, thanks to the dynamic packet header technique, a user can setup different types of firewalls wherever he needs them. For instance, the user can connect to his office from his home using security level 2 and setup his office machine to connect to another server within his organization using a lower security level to gain a better performance. However, if performance is critical and the network is relatively secure, security level 1 can be changed such that no encryption occurs for the packet header.

In order to provide better security, the preferred embodiment allows the user to select if the data should stay in its encrypted form so that only authorized personnel can view the data. This is important for sensitive business data, personnel data, etc. Of course, the key to decrypt the data must be agreed to ahead of time or exchanged over some secured channels to protect the secrecy of the key.

Of course, those skilled in the art will recognize that the user could also have the capability of instructing the system that no encryption will be used. In this case, no encryption would represent a fourth security level (security level 0). Security level 1-3 having been discussed in regard to FIG. 4.

FIGS. 5A-B illustrate the packet queue structure used in the preferred embodiment. FIG. 5A illustrates the TCP/IP and NetBIOS communications structure and FIG. 5B illustrates the modem and RS-232 communications structure. The compressed buffer is a work buffer used to compress data prior to transmission. A packet header is placed at the beginning of the read token and at the beginning of the write token. In the preferred embodiment, the read and write tokens are stored in shared memory.

FIG. 6 illustrates a configuration in which multiple requesters 110 communicate with a single server 602.

FIG. 7 illustrates a configuration in which a single requester 110 communicates with multiple servers 602.

FIG. 8 illustrates a configuration in which a system 802 and multiple servers 804 communicate with one another.

FIG. 9 illustrates a configuration in which multiple systems 802 and multiple servers 804 communicate with one another via modems 124 over phone lines 906 and also over LANs 902 and wide area networks 904. This figure illustrates the ability of the system to interface with multiple communications protocols.

FIG. 10 illustrates a configuration in which multiple requester systems 1002, multiple server systems 1004, and multiple server/requester systems 1006 communicate with one another. The configuration in this figure is similar to that shown in FIG. 9.

FIGS. 11 and 12 illustrate a configuration in a server 1004 which includes communication sessions 1120 to communicate with requesters, encrypter/decrypter 1128, read threads 1114, write threads 1116, packet queues 1110, 1112, a resource control manager 1102 to control user ID, access permission and alias and path storage 1104, 1106, 1108. The cached user ID and access permission 1124 and the cached alias and associated path 1126 caches are used to store data from the access permission storage 1106 and the alias and path storage disks 1108 for improved system performance.

To protect resources on the network domains, an access control list (ACL) is used for each network domain in access permission storage 1106. The ACLs are managed by network administrators to define to which resources a user can access and what kind of accesses the user has to each resource. The system provides a sophisticated ACL so that a user cannot view or access any resources other than those assigned. The following access permissions are used by our ACLs:

```

READ_FILE
WRITE_FILE
CREATE_FILE
DELETE_FILE
EXECUTE_FILE
CHANGE_ATTRIBUTE
ACCESS_SUBDIR
CREATE_SUBDIR
REMOVE_SUBDIR

```

For example, if the user is not permitted access to any subdirectories from a network resource, the user will not see

any subdirectory at all when viewing the network resource. If for some reasons the user knows a particular subdirectory exists under the network resource, he cannot access it anyway. The management of network resources and user access permissions is provided with a user-friendly Graphical User Interface application. Together with the logon procedure, ACLs provide effective protections to the resources on the network domains.

FIG. 12 is a more detailed view of the server 1004 of FIG. 11. A control manager 1122 within the server 1004 is responsible for communication between the server 1004 and other applications on the server 1004 machine. Thus, the server 1004 can be informed if a database has been changed by a resource control application. The server 1004 can also accept a message from another application 102 to send to all or selected clients over active sessions. If an electronic mail system should be needed, the server 1004 can save the message and wait until a client is logged on to send the message over the session. To support these features, the control manager 1122 posts message or e-mail packets to the incoming packet queues 1206 of the sessions 1120. When the server processing threads 1114, 1116 of the sessions 1120 retrieves the packets from the queue 1206, it will process the packets based on the packet types defined in the packet headers.

FIG. 13A-D illustrates the packet headers used in the logon procedure. A session key KS and an initialization vector IV are defined for a communication session between a client and a server 1004 when security level 1 or higher is desired (in security level 0, no encryption is used).

FIG. 13E illustrates a normal packet such as those used during data transfer. When an e-mail or message packet is sent, the preferred embodiment uses security level 2 by default to protect the messages. In security level 2, both packet header and data are encrypted using single  $s_3$ DES encryption.

The requester also has the capability to signal request routers 106 of all applications 102 when a communication session is terminated abnormally whether the request routers 106 are sending request packets or waiting on response packets. In order to perform this feature, the response signals (i.e., the return addresses stored in the request packets) are saved in response-signal queues by the session write thread 1116. Each communication session has a response-signal queue 1206 to reduce the search time. When the response packets are successfully delivered, their corresponding response signals are removed from the queue by the session read threads 1114 of the corresponding communication channels. If an application 102 terminates before its response packets arrive, the response packets are discarded and the response signals are also removed from the queue after all chaining response packets have arrived.

In addition, the read thread of the client session also recognizes different types of packets to determine whether it should route the received packets to the application's request router or to a message manager within the requester. The message manager of the requester is responsible for message and e-mail packets sent from the connected servers. This feature is important because it allows the server to initiate the sending of packets while a session is active. As an example, a hot-link can be defined so that a server can inform the connected clients if a database should be changed or a server administrator can send a message to all or selected clients telling them if a server should be out of service shortly, etc. In a more advanced application, an electronic-mail server application can be written so that the message packets are saved on the server until a client is

13

logged on. At that time, the server will send the saved messages to the connected client.

In the prior art, the requester is the one that translates and formats requests from the applications; thus, it cannot perform preparation ahead of time. In addition, information accumulating in one place could increase the search time. The prior art requires an intrinsic modules in both the application and the requester which may require more resources to be allocated and more machine instructions to be executed. Furthermore, the prior art does not have the capability to accumulate multiple request packets from a requester so that the server can process the next packet request while the previous response packet is traveling back to the requester on the network or being processed by communication devices in their own memory buffers.

In contrast to the prior art, the preferred embodiment contains the formatting and translating code in just one place, the request router 106. The requester only compresses and/or encrypts packet headers and packet data if necessary and then calls the transport functions to send the packets to the server. In addition, requester 110 is also responsible for saving logon and mounting information, managing the communication sessions, and delivering response packets received from multiple network domains to multiple request routers while sending request packets to the multiple network domains. Requester 110 does not need to know the format of the response data, and can deliver the response packets immediately upon receiving them. The request routers 106 can then format or translate the response data in the applications time slices while the requester 110 is waiting for other incoming response packets or reading data from the communication devices 120, 122, 124, 126. Thus, the preferred embodiment achieves better performance than the prior art.

The prior art also requires the intrinsic modules to translate and format the application data from a program stack segment to a parameter block before sending it to its requester where the data is once again formatted or copied into a data communication buffer. In contrast, the request routers 106 in the preferred embodiment format the application data only once and store the formatted data into the write token which will be used by the requester and the communication subsystem to send the request packets to the server. When the response packets arrive, the requester 110 uses the response signals to tell the corresponding request routers that their response packets have arrived. At that time, the request routers 106 transfer response data directly from the read tokens into the application buffers. Thus, the preferred embodiment eliminates the overhead of copying data between memory buffers.

Furthermore, the prior art does not have the dynamic packet header feature to support packet authentication on demand. Neither does its server authenticate the requester to prevent replaying of packets by intruders. The prior art also requires two different programs running on the server to wait for incoming data from different communication protocols. The preferred embodiment only requires the server to be started once for multiple communication protocols.

In general, a session on the server 1004 will support multiple applications on the requester; thus, a server 1004 must somehow remember the resources allocated for the client applications so that these resources can be released whether the client applications terminate abnormally or the communication sessions are destroyed abnormally. Our server supports this feature in each session thread. Since the allocated resources are isolatedly remembered for different requesters, the search time is minimum every time they are

14

added or removed from the memorized list. In addition, security audit can be turned on and off by the network resource manager running on the server over the control channel of the server. The network resource manager can toggle the security audit for users or groups whose IDs are supplied in the auditing request packet, or resources whose names are stored in the auditing request packet. The audit can also be logged based on successful, failed, or both transactions.

In the prior art, the application is the one which determines if a session should be started on the host computer. The application then makes a function call to connect to the host computer and another function call to start a host server process. In the preferred embodiment, the command manager of the requester determines if a connection should be established to couple the client computer to the server computer. Once the connection is established, the server automatically creates a server processing thread to process the client request packets received over the connection. After the connection is established, the command manager also performs the auto-logon itself, not the application. The session can then be shared by all the applications on the client machine.

Thus, the session creation and automatic logon, re-logon or auto-logon are transparent to the applications. If the logon is successful, the server creates a server receiving thread to receive and accumulate request packets in a packet queue so that they will be processed by the server processing thread. When a session disconnect request packet is received, the server receiving and processing threads terminate themselves. However, if the communication session is destroyed abnormally, the server receiving thread simulates a disconnect request packet and appends it to the packet queue to signal the server processing thread to terminate. The server receiving thread then terminates itself.

Note that in the very first logon manually performed by the user, the operation is slightly different than the auto-logon mentioned in the above paragraph. The requester first receives a logon request from the logon application, it establishes the session itself and then performs the logon. This is so done by the command manager of the requester, not by the session manager. The session manager is responsible for dropping the session if no data is transmitted for a certain period of time.

Since request packets are accumulated in the packet queue in the preferred embodiment, the request packets may not be processed immediately upon arrival. In contrast, the prior art must process the request packets immediately to return the status or data to the requester. This may indicate that other applications on the client computer must wait until the return packet has arrived and processed before they can send their requests to the same host computer.

The prior art requires an application to send a function call to the host computer to establish a communication session. Our system establishes a communication session by the requester when it receives a logon request from the logon program or a request router asking for the communication handle. In addition, our server has the capability to reformat and retranslate the request packets in its own request router before forwarding them to the requester located on the server when the network resources do not reside on the server. That is, multiple servers can be connected together as shown in FIGS. 7-10 to expand the amount of network resources available to requesters. Note that this feature requires the intermediate servers administrator(s) to manually logon the designated servers since the logon passwords are not stored on the intermediate servers. Users on request-

ers can perform this logon remotely if their access permissions in the ACLs of the intermediate servers indicate that they can execute programs on the intermediate servers. However, caution must be taken and security level 3 is advised when using this feature since logon user IDs and passwords must be sent along with the executing request packets.

As shown earlier, the very first logon packet is encrypted with three different keys for different parts of the packet. The header of the logon packet is encrypted with a key generated from the server name. This is design to detect outside intruders early in the verification process. For intruders working inside an organization, the server name may be known. Then it comes the middle part of the logon packet which contains the 64-bit random number and the CRC value. These are the heart of the verification since it is encrypted with the key generated from the user ID and the secret password. This scheme allows the server to detect the intruding logon right on the very first packet. The challenge-response process that following the logon packet is to defeat re-played packets.

The encryption system used in the preferred embodiment has several other advantages, as follows. The long term key is derived from a user ID and a secret password. It has 192 bits and is used in a  $s^3$ DES encryption enhanced with Cipher Block Chaining (CBC) mode. The short term key is generated with a formula similar but more secure than the one suggested in X9.17 and changed every time a session is established between two nodes on the network. Thus, the encryption occurs at the application layer which exposes the source and destination addresses of the packets when used with TCP/IP and NetBIOS protocols but the intruders must deal with different keys whose lengths are either 40, 112 or 240 bits for different pair of nodes on the network. In addition, the short term key is encrypted and only sent once when the communication session between two nodes is established, not in every packet; thus, it reduces the traffic between two nodes.

Furthermore, the prior art only protects data between site-firewalls, not between nodes. In many cases, data must be protected between nodes within an organization. For instance, high-rank management officers within a private network may want to exchange restricted confidential information without leaks to their employees.

Encryption at the application layer also reduces the cost of replacing the existing network layer and can be done on demand when protection to data is needed. Different security firewalls can easily be established between any pair of nodes with a single click of the fingertip.

FIGS. 14A-C and 15A-E illustrate an alternative preferred embodiment of the invention. FIGS. 14A-C illustrate the memory layout of the packets used in the alternative preferred embodiment. FIGS. 14A-C differ from the memory layout shown in FIGS. 4A-C in that the "PKT VERSION FIELD" has been deleted. This reduces the amount of data to be transferred in the packets which in turn reduces storage requirements and improves performance. FIG. 15A-D illustrate the packet structures used in the logon procedure of the alternative preferred embodiment. A session key KS and an initialization vector IV are defined for a communication session between a client and a server 1004 when security level 1 or higher is desired.

FIG. 15B illustrates a normal packet structure such as that used during data transfer, when a  $s^3$ DES algorithm is in use. Its S-boxes are cryptographically modified and selected based on a given key. In the preferred embodiment, the random numbers Ra and Rb are XOR'ed to by both client

and server to generate a key for the S-boxes. Thus, security of data exchanged over the communication channel is not only dependent to the session encryption key but also to the S-box key.

In FIG. 15A, the 64 bit R field in the beginning of the header includes a 32 bit constant which identifies the source system of the packet. The R field is used as a key to encrypt the 16 byte field which holds the User ID. For use in international communications, the high order 24 bits can be set to zero. By so doing, the level of encryption security can be reduced to comply with United States laws governing encrypted communications.

The logon procedure facilitates a three-way authentication to prevent the password from being transferred over the network and allows both client and server to authenticate each other as well as detecting packet replaying. First, the client generates a 32-bit random number V concatenated by a predefined 32-bit constant to form a 64-bit number R. This 64-bit random R is used as a  $s^3$ DES key to encrypt the user ID to make the user ID look random for every logon packet. Then the client generates a 64-bit key K from the server name to encrypt the 64-bit value R. Now, the client generates a 192 bit key Ka from the user ID and password using a one-way hash function such as the Secure Hash Standard (SHS) specified in the Federal Information Processing Standards Publication 180 (FIPS PUB 180). It then generates a 64-bit random number Ra, calculates the CRC signature C whose initial value is the random number V and consisting of the random number Ra, the original user ID, and the random number R. The client then encrypts the random number Ra and the signature C using the key Ka. Thus, the very first logon packet is encrypted with three different keys for different parts of the packet (see FIG. 13A-D).

When the server receives the first logon request packet, it generates a key K2 from its machine name and the SHS to decrypt the packet header for verification. If the packet header does not contain the predefined constant, the user has selected a wrong server, i.e., the user tries to access the server over the phone line but does not know the server name since the phone number is a public record but the server name is a private one. However, if the packet header contains valid data, the server uses the decrypted 64-bit packet header as a key to decrypt the user ID. The user ID is then used to search a database for an access record. If the access record cannot be found, the user has entered an invalid ID and the session is terminated. However, if the access record is found, the server verifies if the user is allowed to access the network resources on this date and at this time.

After the access date and time are verified, the server will retrieve the associated one-way-hashed password Kb from an encrypted password file to decrypt the random number Ra and the CRC signatures. The key Kk used to decrypt the password file is selected by the server administrator at installation. The key must be entered every time the server process is started since it is not stored on the system.

Now, the first logon packet is decrypted. The server calculates the CRC signature of the random number Ra, the user ID, and the random number R. If the calculated signatures match with the decrypted signatures C stored in the packet, the server manipulates the client random number Ra with a predefined formula to form Ra', generates a random number Rb, and encrypts both random numbers Ra' and Rb with the password Kb before sending the second logon packet to the client.

After the client decrypts the first logon response packet, it manipulates the random number Ra with the predefined

formula and compares it against the one returned from the server. If the numbers match, the client knows that it communicates with a correct server, not a fraud server where an eavesdropper has set up to echo back captured packets. The client now manipulates the random number Rb with another predefined formula to form Rb' and concatenates it with the client's initiating data, i.e., the client initial packet sequence number and the encryption mode for the session, to form a second logon packet. The client then masks the initial data with the random number Ra to hide the possibly known text, encrypts the third logon packet and sends it to the server. Note that the client starts to send its initial data to the server only if the random number Ra' is verified to ensure that the server is a trusted party.

After decrypting the third logon packet, the server also manipulates the random number Rb with the same predefined formula used by the client and verifies if the random numbers are matched to assure that it communicates with a correct client. If the random numbers match, the server knows that it is communicating with an authorized client and the first logon packet was not a replayed packet. The server then saves the client initiating data, generates a session key Ks and an initialization vector IV with the formula similar but more secure than the one specified in Appendix C of the ANSI X9.17 standard, encrypts and sends them to the client. Along with the session key Ks and the IV, the server also sends its initiating data (i.e., the server initial packet sequence number, approved encryption or compression method). The client and server initial packet sequence numbers are used to detect packet deletion and insertion for data exchanged after the logon procedure. The server initial data is also masked with the client random number Ra to hide possibly known text. Note also that the server only sends its initial data after the number Rb is verified to ensure the client is a valid party random number.

After decrypting the third logon packet, the client now saves the session key Ks and the IV in its own memory for future communication with the server. To improve performance, the client and server build encryption and decryption key schedules of s<sup>3</sup>DES and stores them in the session information block structure as shown in FIGS. 5A-B. The S-boxes of s<sup>3</sup>DES are also combined with the P-box of DES algorithm to further enhance encryption speed. As described so far, the authentication techniques not only provide a secure identification procedure, but also a secure negotiation protocol to set up communication sessions (i.e., encryption method, compression method, operating platform, language, etc.).

As appreciated, the following symbols may help to clarify the logon procedure:

1. A→B: EK(R)+EKa(Ra,f(Ra,R,UID))+ER(UID)
2. A←B: EKb(Ra',Rb)
3. A→B: EKa(Rb',Dc)
4. A←B: EKb(IV,Ks,Ds)

where:

- A—a requester
- B—a server
- E—a symmetric cryptosystem such as DES or s<sup>3</sup>DES
- K—an encryption key generated from the server name
- R—a 32-bit random number concatenated by a predefined constant
- Ka—a 192-bit key one-way hashed from the user ID and password
- Ra—a 64-bit random value generated by A
- f( )—a hash function such as CRC to calculate the signature
- UID—a user ID
- Kb—a 192-bit one-way hashed key retrieved from a database
- ha( )—a hash function to manipulate the random number Ra
- Rb—a 64-bit random value generated by B
- hb( )—a hash function to manipulate the random number Rb
- Dc—client initial data masked with Ra
- IV—an initial chaining vector for encryption in CBC mode
- Ks—a session encryption key
- Ds—server initial data masked with Ra
- Ra'—ha(Ra)
- Rb'—hb(Rb)

After the logon procedure is completed, the random numbers Ra and Rb are XOR'ed by both the client and server to generate a key for the S-boxes used by DES or s<sup>3</sup>DES. The key-dependent S-boxes will make DES and s<sup>3</sup>DES harder to cryptanalyze since the key length is longer. Thus, every communication session will have a different session key and different S-boxes to be used by DES and s<sup>3</sup>DES in addition to the IV.

Variants of the S-boxes, such as s<sup>3</sup>DES S-boxes can be used in place of DES S-boxes to provide improved security. An example of such S-boxes are included in Table 1, below. In table 1, the s<sup>3</sup>DES S-box 1 and S-box 2 are reversed from an original s<sup>3</sup>DES S-box configuration.

TABLE 1

<b>S-box 1:</b>															
13	14	0	3	10	4	7	9	11	8	12	6	1	15	2	5
8	2	11	13	4	1	14	7	5	15	0	3	10	6	9	12
14	9	3	10	0	7	13	4	8	5	6	15	11	12	1	2
1	4	14	7	11	13	8	2	6	3	5	10	12	0	15	9
<b>S-box 2:</b>															
15	8	3	14	4	2	9	5	0	11	10	1	13	7	6	12
6	15	9	5	3	12	10	0	13	8	4	11	14	2	1	7
9	14	5	8	2	4	15	3	10	7	6	13	1	11	12	0
10	5	3	15	12	9	0	6	1	2	8	4	11	14	7	13
<b>S-box 3:</b>															
13	3	11	15	14	8	0	6	4	15	1	12	7	2	10	9

TABLE 1-continued

4	13	1	8	7	2	14	11	15	10	12	3	9	5	0	6
6	5	8	11	13	14	3	0	9	2	4	1	10	7	15	12
S-box 4:															
9	0	7	11	12	5	10	6	15	3	1	14	2	8	4	13
5	10	12	6	0	15	3	9	8	13	11	1	7	2	14	4
10	7	9	12	5	0	6	11	3	14	4	2	8	13	15	1
3	9	15	0	6	10	5	12	14	2	1	7	13	4	8	11
S-box 5:															
5	15	9	10	0	3	14	4	2	12	7	1	13	6	8	11
6	9	3	15	5	12	0	10	8	7	13	4	2	11	14	1
15	0	10	9	3	5	4	14	8	11	1	7	6	12	13	2
12	5	0	6	15	10	9	3	7	2	14	11	8	1	4	13
S-box 6:															
4	3	7	10	9	0	14	13	15	5	12	6	2	11	1	8
14	13	11	4	2	7	1	8	9	10	5	3	15	0	12	6
13	0	10	9	4	3	7	14	1	15	6	12	8	5	11	2
1	7	4	14	11	8	13	2	10	12	3	5	6	15	0	9
S-box 7:															
4	10	15	12	2	9	1	6	11	5	0	3	7	14	13	8
10	15	6	0	5	3	12	9	1	8	11	13	14	4	7	2
2	12	9	6	15	10	4	1	5	11	3	0	8	7	14	13
12	6	3	9	0	5	10	15	2	13	4	14	7	11	1	8
S-box 8:															
13	10	0	7	3	9	14	4	2	15	12	1	5	6	11	8
2	7	13	1	4	14	11	8	15	12	6	10	9	5	0	3
4	13	14	0	9	3	7	10	1	8	2	11	15	5	12	6
8	11	7	14	2	4	13	1	6	5	9	0	12	15	3	10

The generated S-boxes can further be combined by the P-Box of the DES algorithm specified in FIPS 41 to speed up the encryption process. These condensed SP-Boxes are stored in the memory as shown in FIGS. 5A and 5B.

The s<sup>3</sup>DES encryption method disclosed above can be used in several alternative embodiments, each of which provides distinct advantages. For example, each client in a system can have its own set of s<sup>3</sup>DES S-boxes so that data is encrypted differently for each session. Therefore, even if the s<sup>3</sup>DES S-box encryption table data was deciphered for one client, the other clients would still be protected because their encrypted data are different.

Another method of improving security is accomplished by storing the s<sup>3</sup>DES S-box encryption table in alterable storage, such as system memory, PROMs, EPROMs, etc. The client and server can selectively update the s<sup>3</sup>DES S-box encryption table data between communication sessions or when otherwise convenient. An advantage associated with this technique is that since the system can change the s<sup>3</sup>DES S-box encryption table between communication sessions, the client is protected on subsequent logons from intruders who deciphered the s<sup>3</sup>DES S-box encryption table from the previous session.

There are both legal restrictions and costs associated with the use of encryption codes. For example, U.S. law prohibits encryption techniques which are difficult to break from being exported to foreign countries. Likewise, the longer the encryption code, the more resources the code requires to decipher. By masking the leading bits in the session key to zero, the code can effectively be altered to shorter lengths. In addition to complying with U.S. law, this also allows the server to provide different security levels to different clients. For example, a domestic client may have a higher level code for data transmission within the United States while a foreign client would have a lower security level due to a masking of bits in the session key.

The server can examine the source system identification provided by the client during the logon procedure. If the

client requests an unauthorized security level, the server can reset the security level to a permitted level of security for that particular client.

Other performance enhancements can be realized by sending multiple requests from a client system to a server in a single transmission. In addition, system performance can also be improved by preventing subsequent packet data from being transmitted between a server and a client until the previous packet data has been responded to. Requestors can batch application work within the client machine and communicate with the remote server resulting in remote batching. As a result, the requestor acts as a remote server function. This function is done asynchronously to enhance performance.

Finally, the communication subsystem of the preferred embodiments are a foundation for multiple applications when their uses are in demand. With just one communication session between a client and a server, packet sending can be initiated by either party to conduct file transfers, broadcast messages, or E-mail messages. In addition to minimum resources and maximum performance, security is also provided to protect the secret of the data.

The preferred embodiment envisions a security system based on s<sup>3</sup>DES S-box encryption, and for ease of discussion, s<sup>3</sup>DES S-box encryption was used in the foregoing discussion. However, alternative S-box encryption methods, such as DES S-box encryption, can be substituted.

While the invention has been described with respect to a preferred embodiment thereof, it will be understood by those skilled in the art that various changes in detail may be made therein without departing from the spirit, scope, and teaching of the invention. For example, the size of encryption keys can be changed, algorithms used to generate the encryption keys can be changed, the device can be implemented in hardware or software, etc. Accordingly, the invention herein disclosed is to be limited only as specified in the following claims.



I claim:

1. A method of securely transmitting packet data between a client and a server with packets encrypted by S-box data, including the steps of:

- using at least one communication channel to transmit packets between at least one client and a server;
- encrypting in the client a first logon packet including information identifying the client source system and transmitting the first logon packet to the server;
- decrypting the first logon packet in the server;
- encrypting a second logon packet in the server with client authenticating information and transmitting the second logon packet to the client;
- decrypting the second logon packet in the client;
- encrypting in the client a third logon packet with session information and transmitting the second logon packet to the server;
- decrypting the third logon packet in the server;
- encrypting a fourth logon packet in the server with session information and transmitting the fourth logon packet to the client; and
- decrypting the fourth logon packet in the client;
- transmitting encrypted data packets between the client and server which are encrypted using S-box encryption; whereby the client and server can establish secure communications by bi-directionally transmitting encrypted data.
- 2. A method, as in claim 1, wherein a plurality of clients are communicating with the server and each client's data packets are encrypted with a different S-box table data.
- 3. A method, as in claim 2, wherein a new S-box is selected for a client each time the client establishes a connection with the server.
- 4. A method, as in claim 3, including the further steps of:
  - storing the S-box encryption table data in updatable storage;
  - selectively altering the S-box data.
- 5. A method as in claim 3, including the further steps of:
  - using the S-box table in updatable storage to encrypt data;
  - selectively altering the S-box data with a key chosen by both client and server.
- 6. A method, as in claim 1, including the further steps of:
  - storing the S-box encryption table data in updatable storage;
  - selectively altering the S-box data.
- 7. A method, as in claim 1, including the further steps of:
  - using at least two selectable encryption schemes, including at least a first encryption scheme for a first security level and at least a second encryption scheme for a second security level; and
  - the server determines if the security level requested by a client is authorized and if the security level is unauthorized, the server reduces the security level to a permissible level.
- 8. A method, as in claim 7, including the further steps of:

using the S-box table in updatable storage to encrypt data; selectively altering the S-box data with a key chosen by both client and server.

9. A method, as in claim 7, wherein a plurality of clients are communicating with the server and each client's data packets are encrypted with different S-boxes.

10. A method, as in claim 9, wherein a new S-box is selected for a communication session by both the client and the server each time the client establishes a connection with the server.

11. A method, as in claim 10, including the further steps of:

storing the S-box encryption table data in updatable storage;

selectively altering the S-box data.

12. A method, as in claim 10, including the further steps of:

using the S-box table in updatable storage to encrypt data;

selectively altering the S-box data with a key chosen by both client and server.

13. A method, as in claim 9, wherein data packets from multiple clients are sent to the server in a single transmission.

14. A method, as in claim 13, wherein a new  $s^3$ DES S-box is selected for a communication session by both the client and the server each time the client establishes a connection with the server.

15. A method, as in claim 14, wherein succeeding data packets from a client are continuously sent to the server via a write thread and responses from the server are continuously received via a read thread.

16. A method, as in claim 15, including the further steps of:

using the S-box table in updatable storage to encrypt data;

selectively altering the S-box data with a key chosen by both client and server.

17. A method, as in claim 9, wherein succeeding data packets from a client are continuously sent to the server via a write thread and responses from the server are continuously received via a read thread.

18. A method, as in claim 17, including the further steps of:

storing the S-box encryption table data in updatable storage;

selectively altering the S-box data.

19. A method, as in claim 17, including the further steps of:

using the S-box table in updatable storage to encrypt data;

selectively altering the S-box data with a key chosen by both client and server.

20. A method, as in claim 1, including the further steps of:

using the S-box table in updatable storage to encrypt data;

selectively altering the S-box data with a key chosen by both client and server.

\* \* \* \* \*



US006505230B1

(12) **United States Patent**  
Mohan et al.

(10) **Patent No.:** US 6,505,230 B1  
(45) **Date of Patent:** \*Jan. 7, 2003

(54) **CLIENT-SERVER INDEPENDENT INTERMEDIARY MECHANISM**

(75) **Inventors:** Sudhir Mohan, Santa Clara; Umesh R. Patil, San Jose; Daniel S. Jordan, San Francisco, all of CA (US)

(73) **Assignee:** Pivta, Inc., Cupertino, CA (US)

(\*) **Notice:** This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) **Appl. No.:** 09/312,308

(22) **Filed:** May 14, 1999

(51) **Int. Cl.<sup>7</sup>** ..... G06F 13/00

(52) **U.S. Cl.** ..... 709/202

(58) **Field of Search** ..... 709/200, 201, 709/202, 203, 204; 707/1, 3, 4, 507

(56) **References Cited**

**U.S. PATENT DOCUMENTS**

5,974,430 A \* 10/1999 Mutschler, III et al. .... 707/505  
6,085,239 A 7/2000 Kubo et al.  
6,192,380 B1 \* 2/2001 Light et al. .... 707/505

**FOREIGN PATENT DOCUMENTS**

JP 10-187525 7/1998  
WO WO 98/26344 6/1998  
WO WO 00/65773 11/2000

**OTHER PUBLICATIONS**

"A Bookmarking Service For Organizing and Sharing URL's", R. Keller, S. Wolfe, J. Chen, J. Rabinowitz, N. Mathe, White Paper, Computer Networks and ISDN Systems Issue 29, pp. 1104-1114, 1997.

"WWW Assisted Browsing by Reusing Past Navigations of a Group of Users", M. Jaczynski and B. Trousse, Advances in Case-Based Reasoning, DE, Springer Verlag, Sep. 1998, pp. 160-171.

"Broadway, A Case-Based Browsing Advisor for the Web", M. Jaczynski and B. Trousse, Advances in case-based reasoning, DE, Springer Verlag, Sep. 1998, pp. 692-698.

"Intermediaries: New Places for Producing and Manipulating Web Content", R. Barrett and P. Maglio, Computer Networks and ISDN Systems Issue 30, pp. 509-518, 1998.

"Roaming Access Now Available", Apr. 21, 1999, Computing News, www.cns.yorku.ca/archive/compNews/Articles/9904/roam.htm.

Netscape: "Flexible Roaming Access" White Paper, www.home.netscape.com/communicator/v4.5/whitepaper/roaming.html, Jan. 17, 2001.

\* cited by examiner

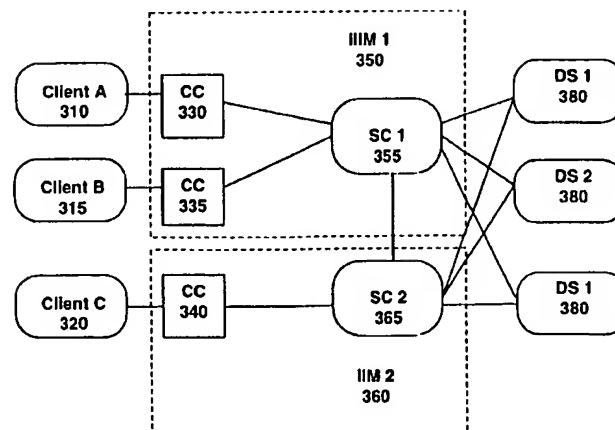
*Primary Examiner*—Robert B. Harrell

(74) *Attorney, Agent, or Firm*—Blakely Sokoloff Taylor & Zafman

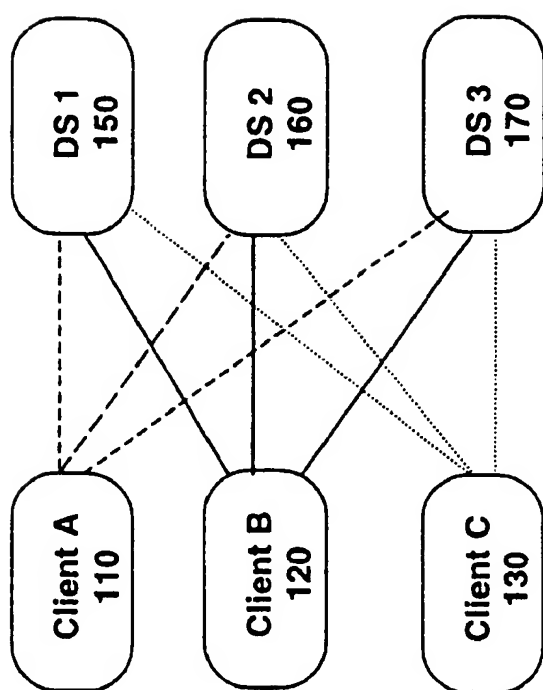
(57) **ABSTRACT**

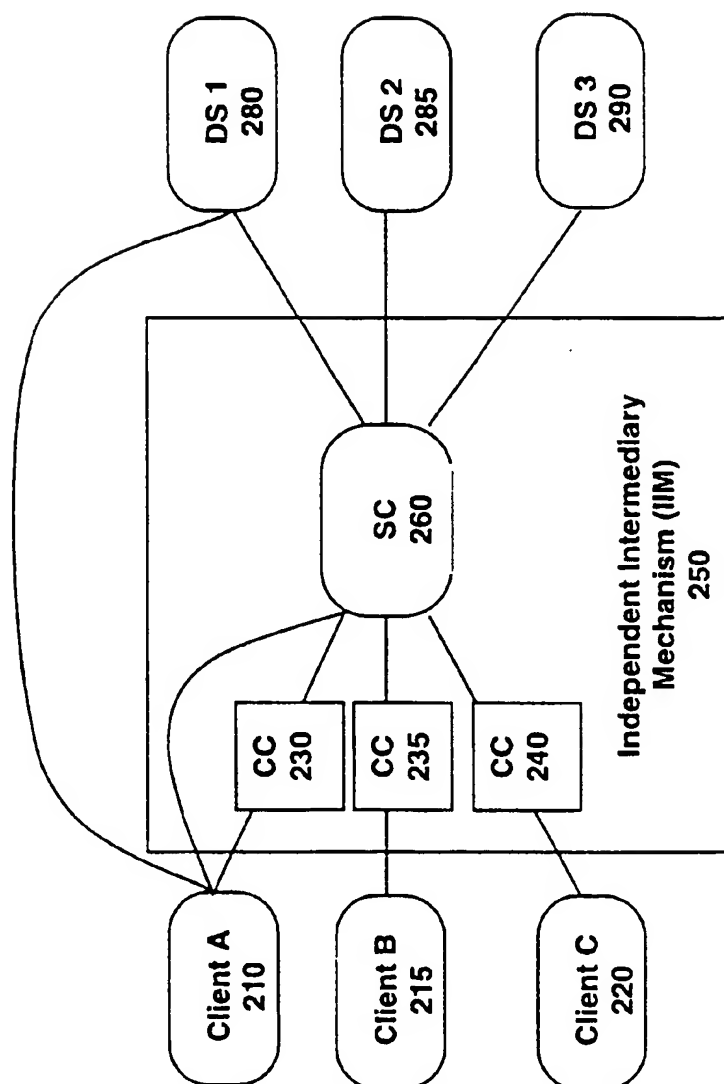
A method and apparatus for a client-server independent intermediary mechanism is provided. The method comprises displaying a frame including a user interface of the IIM, the frame framing a destination server display area (DSDA). The method further comprises retrieving data for display from a destination server, and instrumenting the data prior to display such that future data retrieved from the destination server is displayed in the DSDA, without writing over the frame.

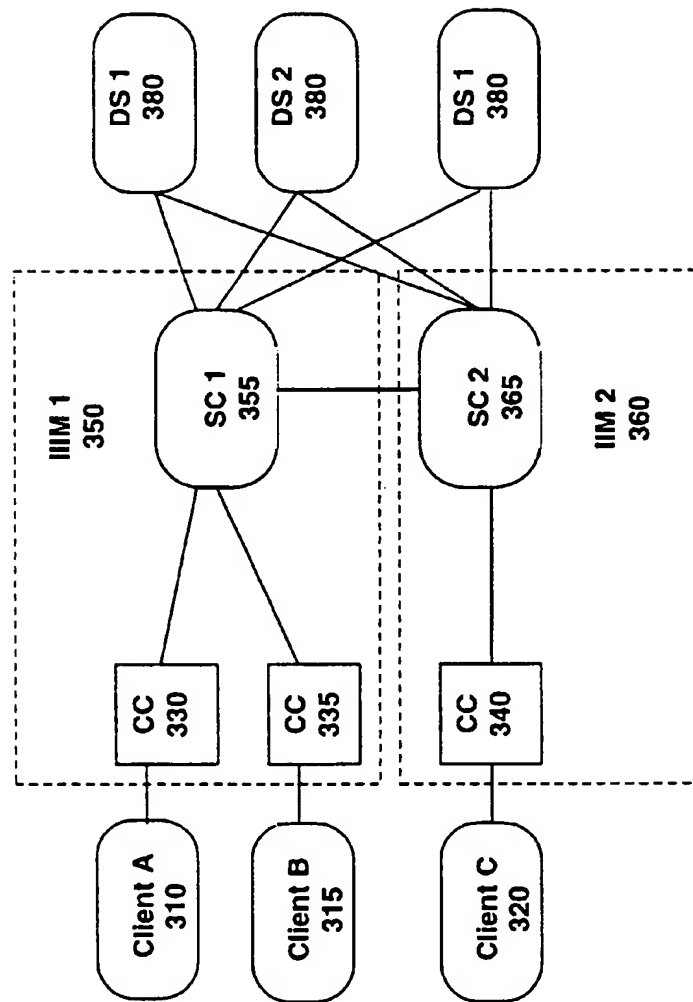
**21 Claims, 25 Drawing Sheets**





**Fig. 1 (Prior Art)**

**Fig. 2**

**Fig. 3A**

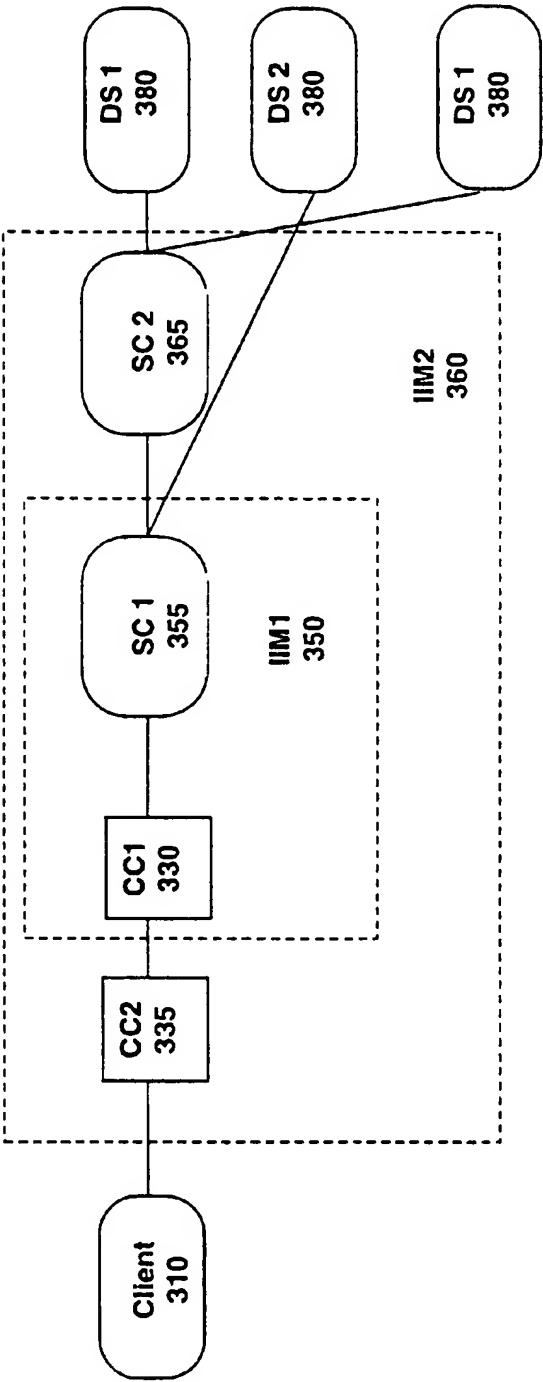


Fig. 3B

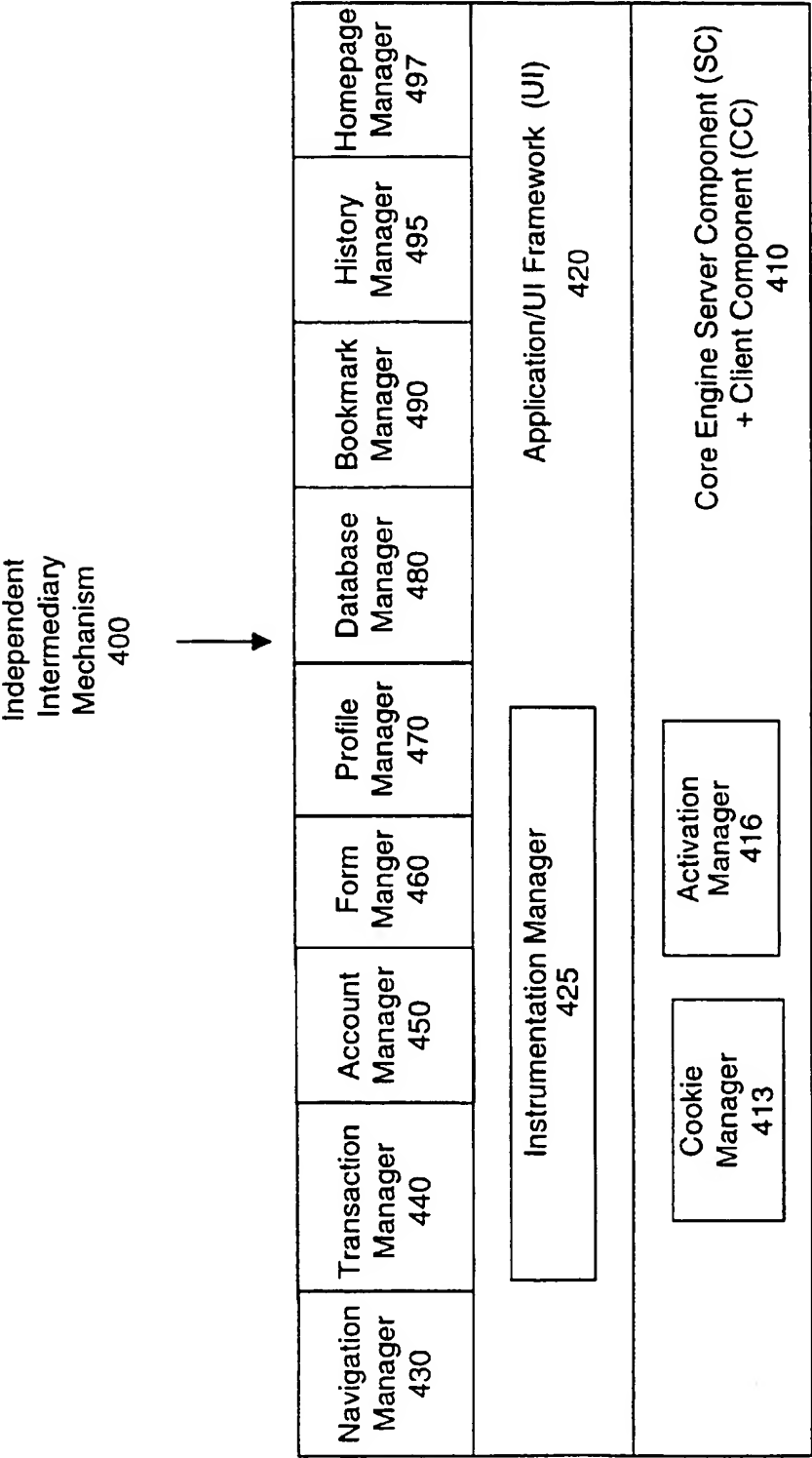


Fig. 4

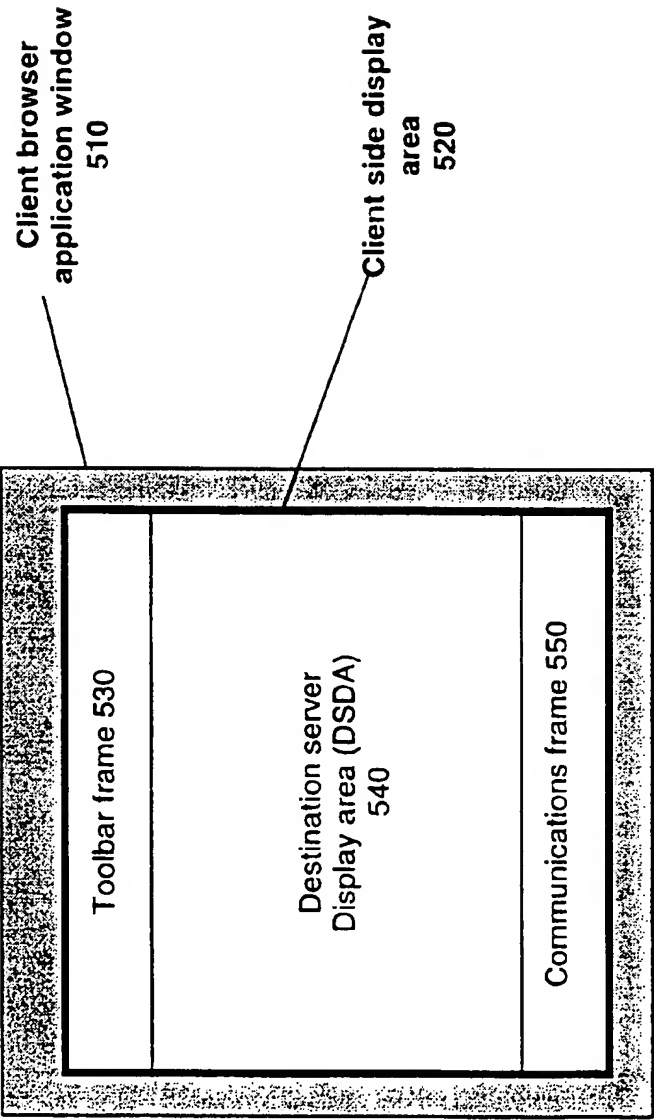


Fig. 5

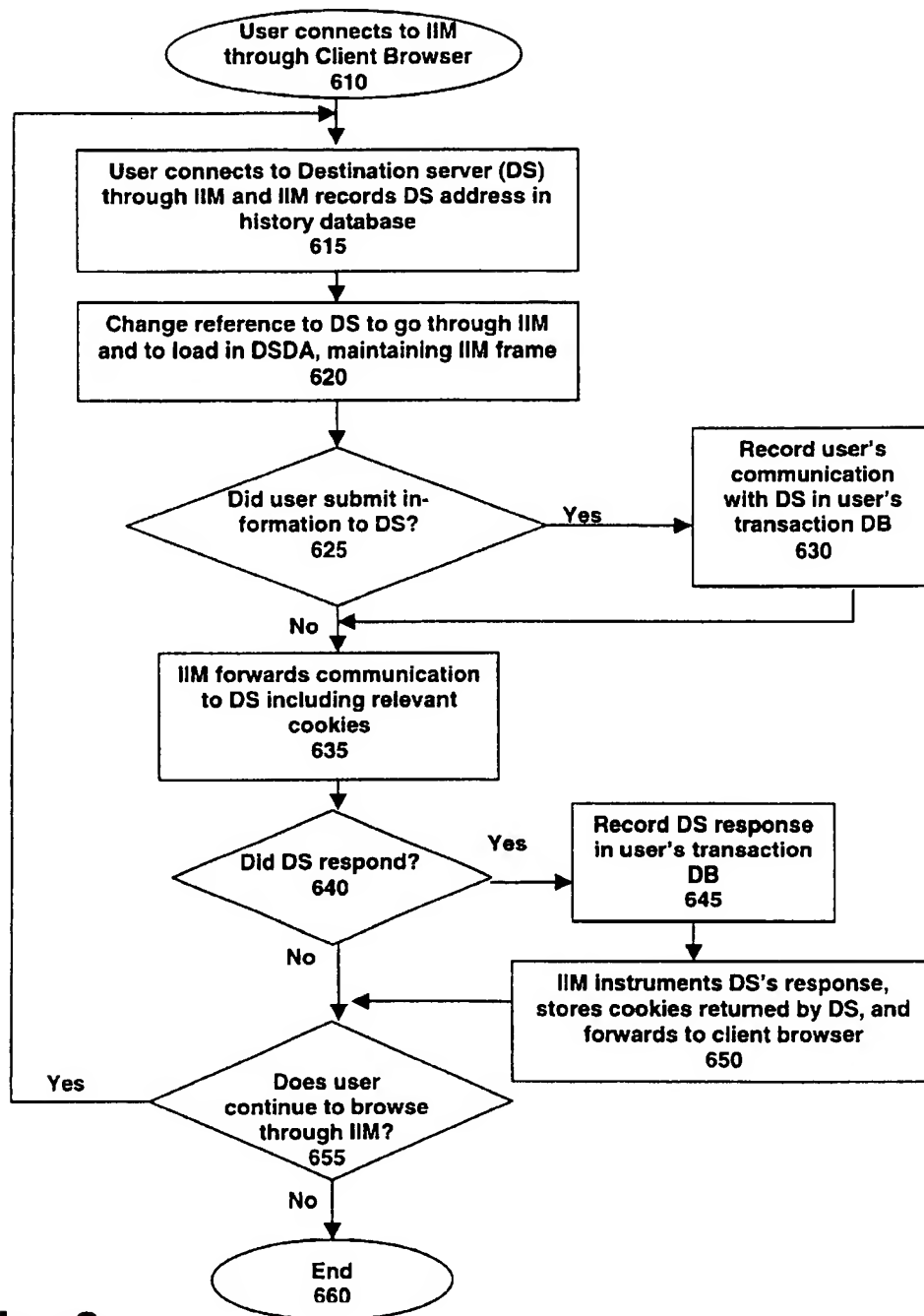


Fig. 6

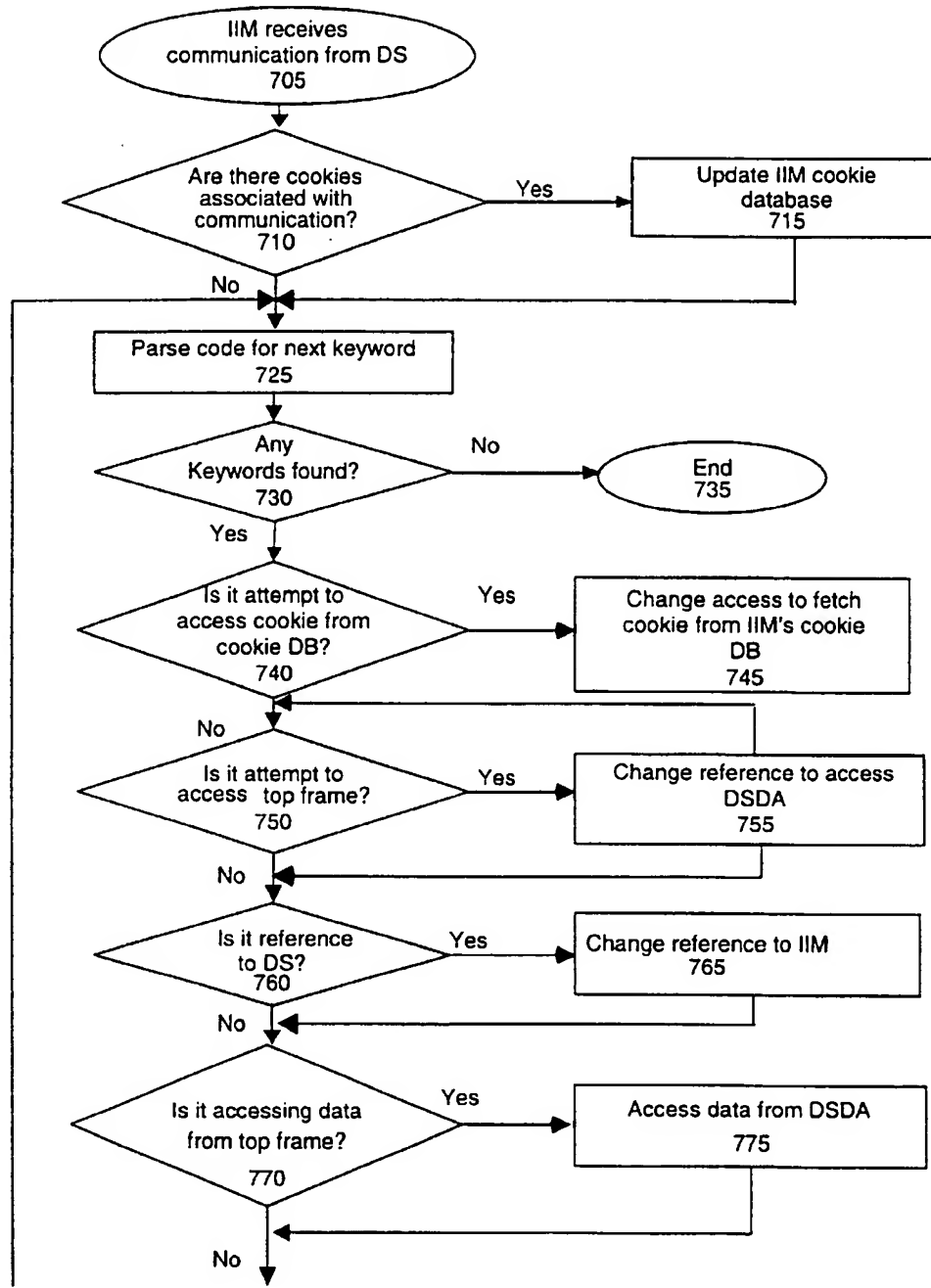
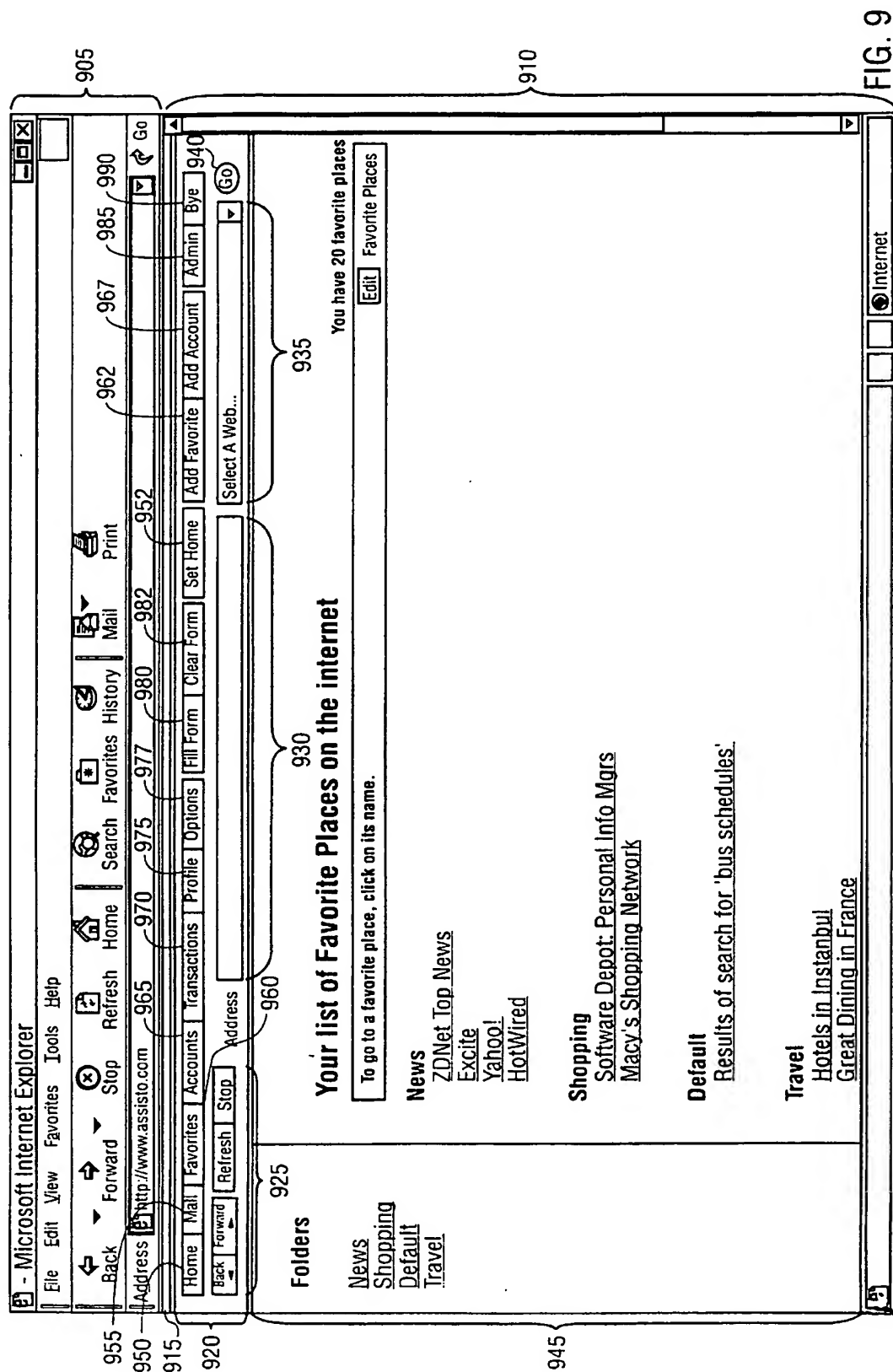


Fig. 7







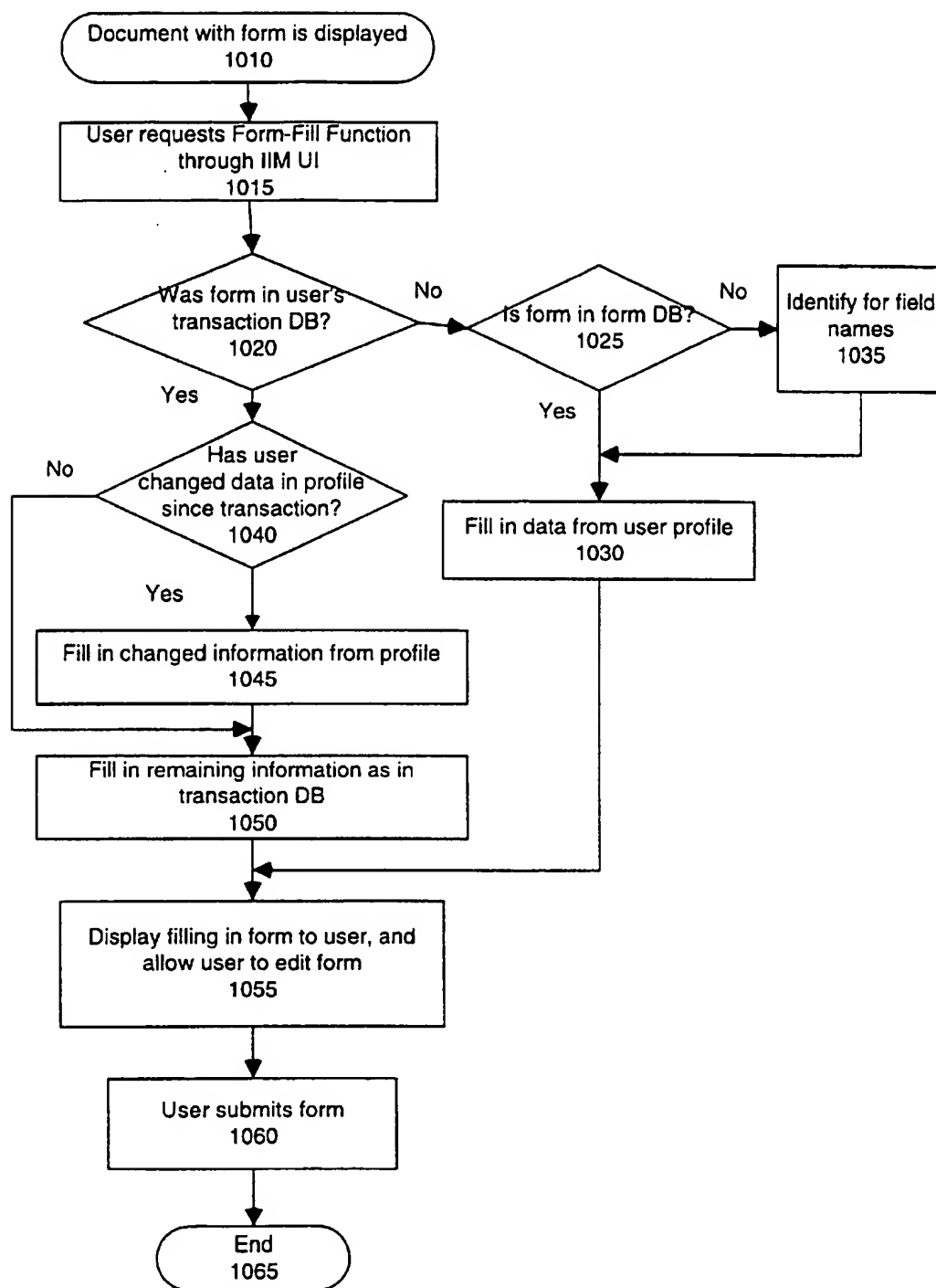
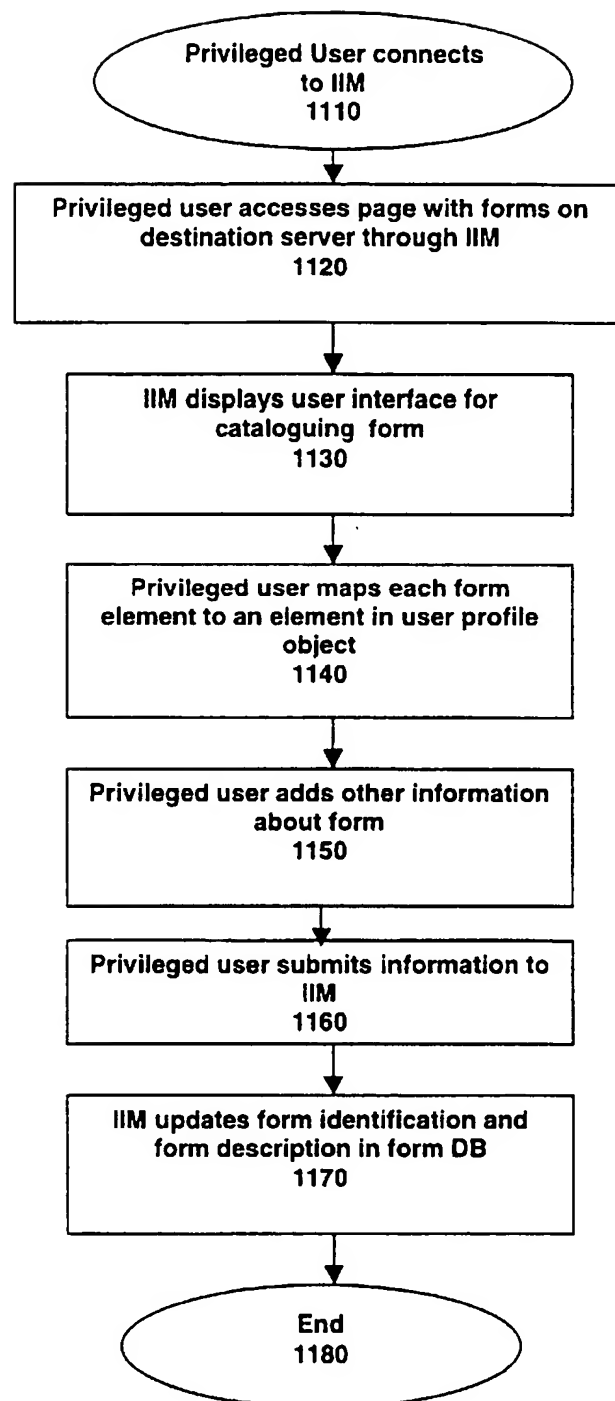
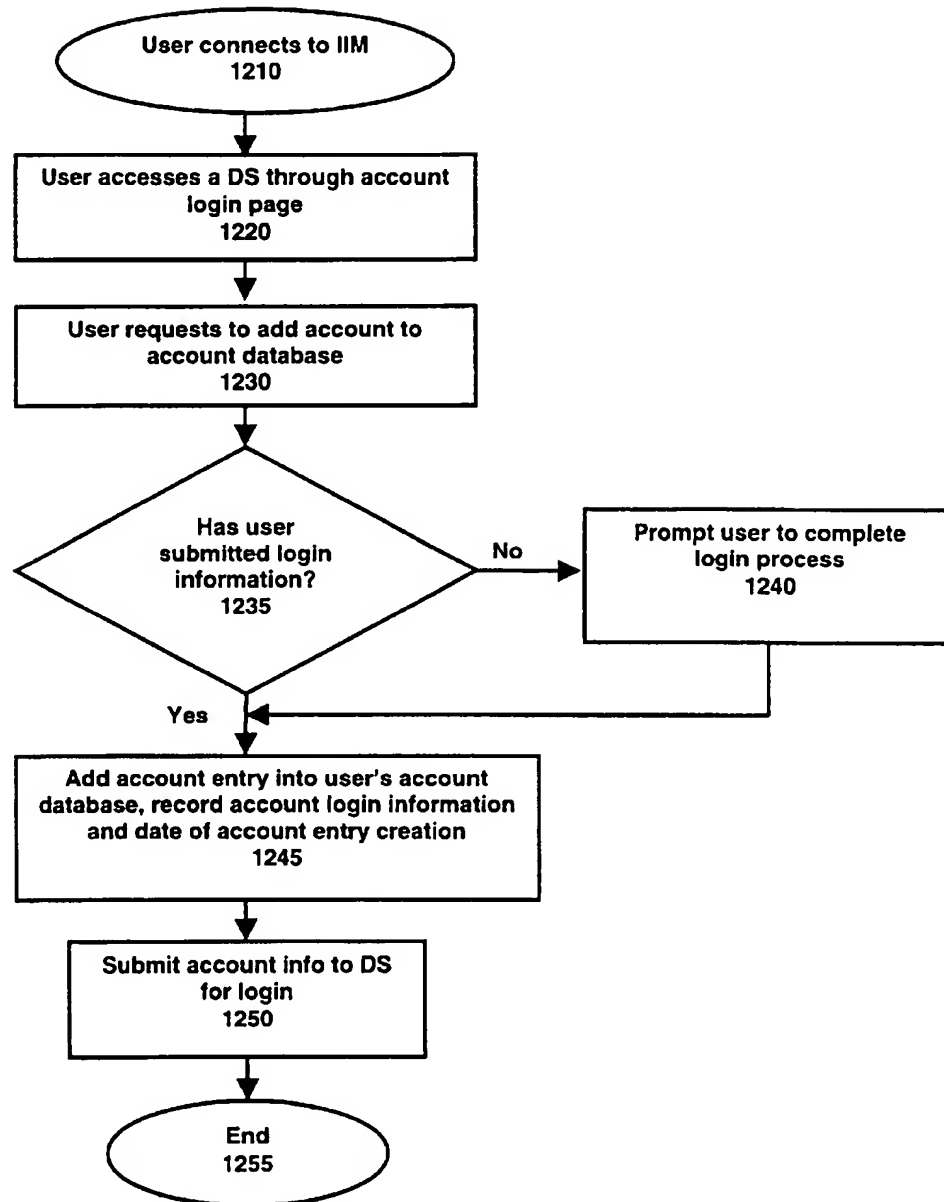
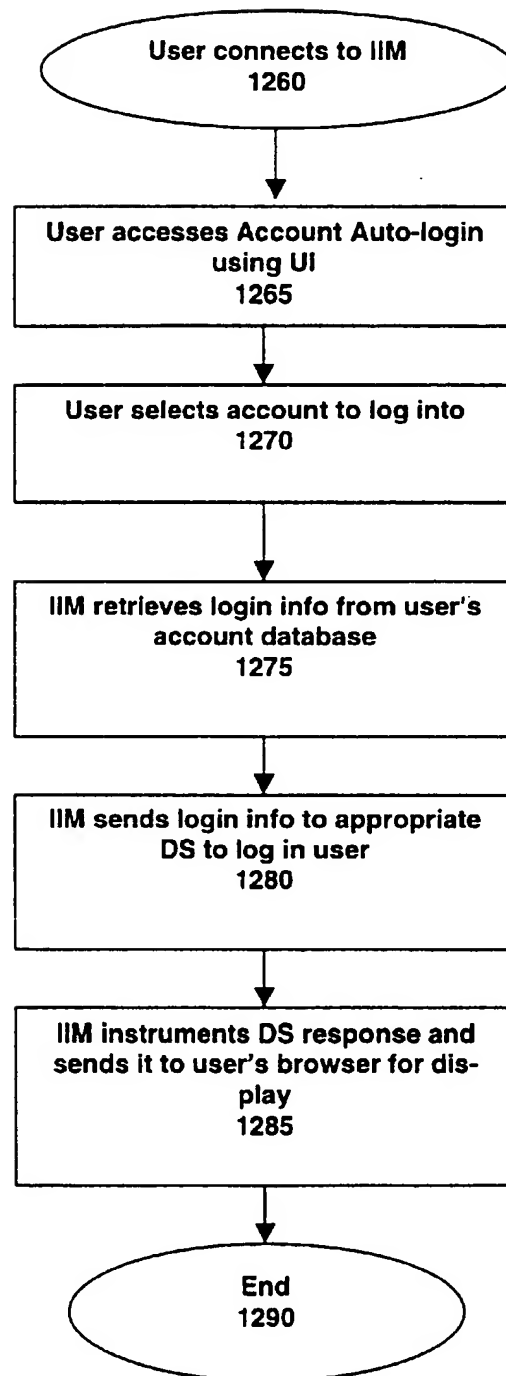


Fig. 10

**Fig. 11**

**Fig. 12A**

**Fig. 12B**

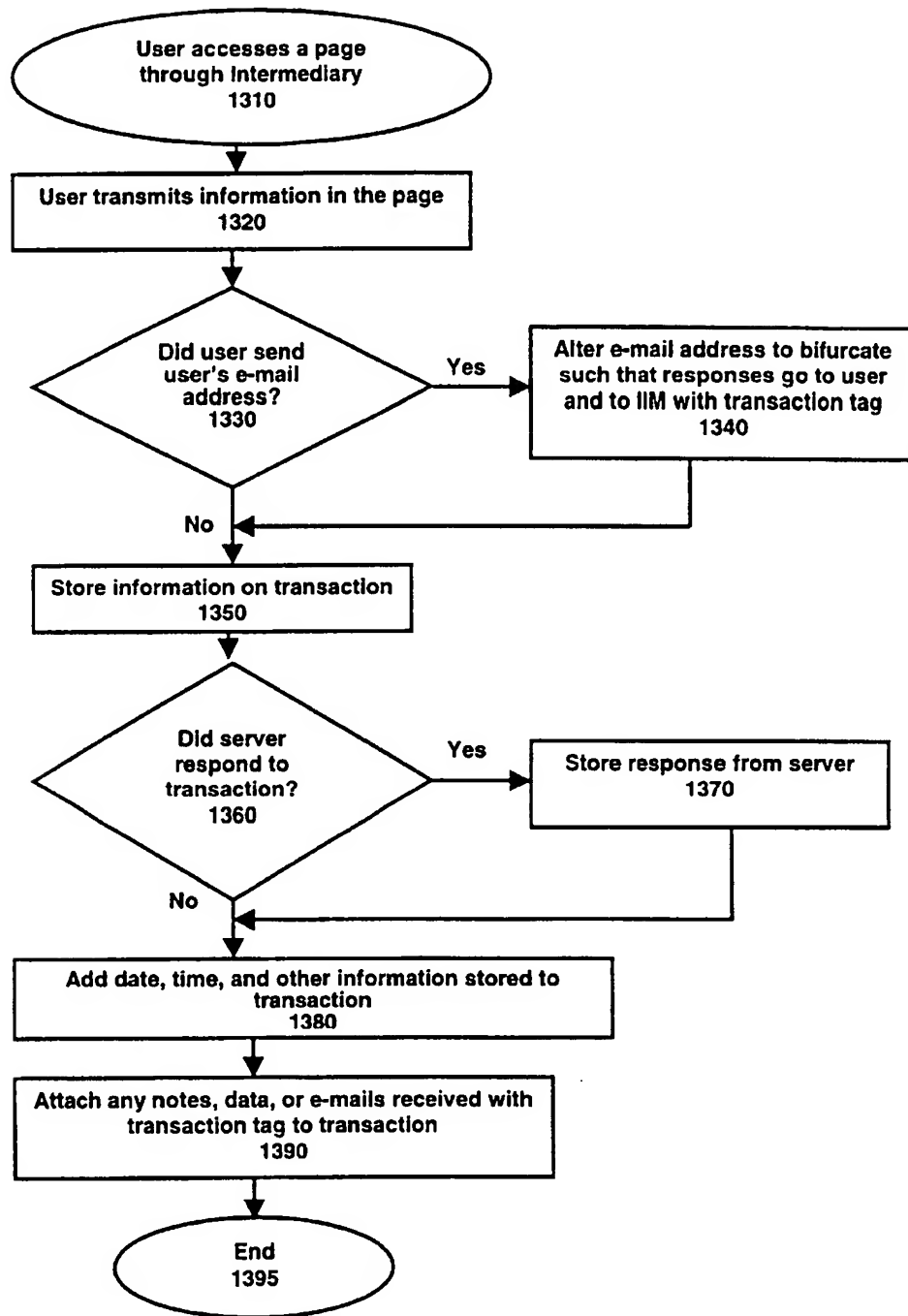
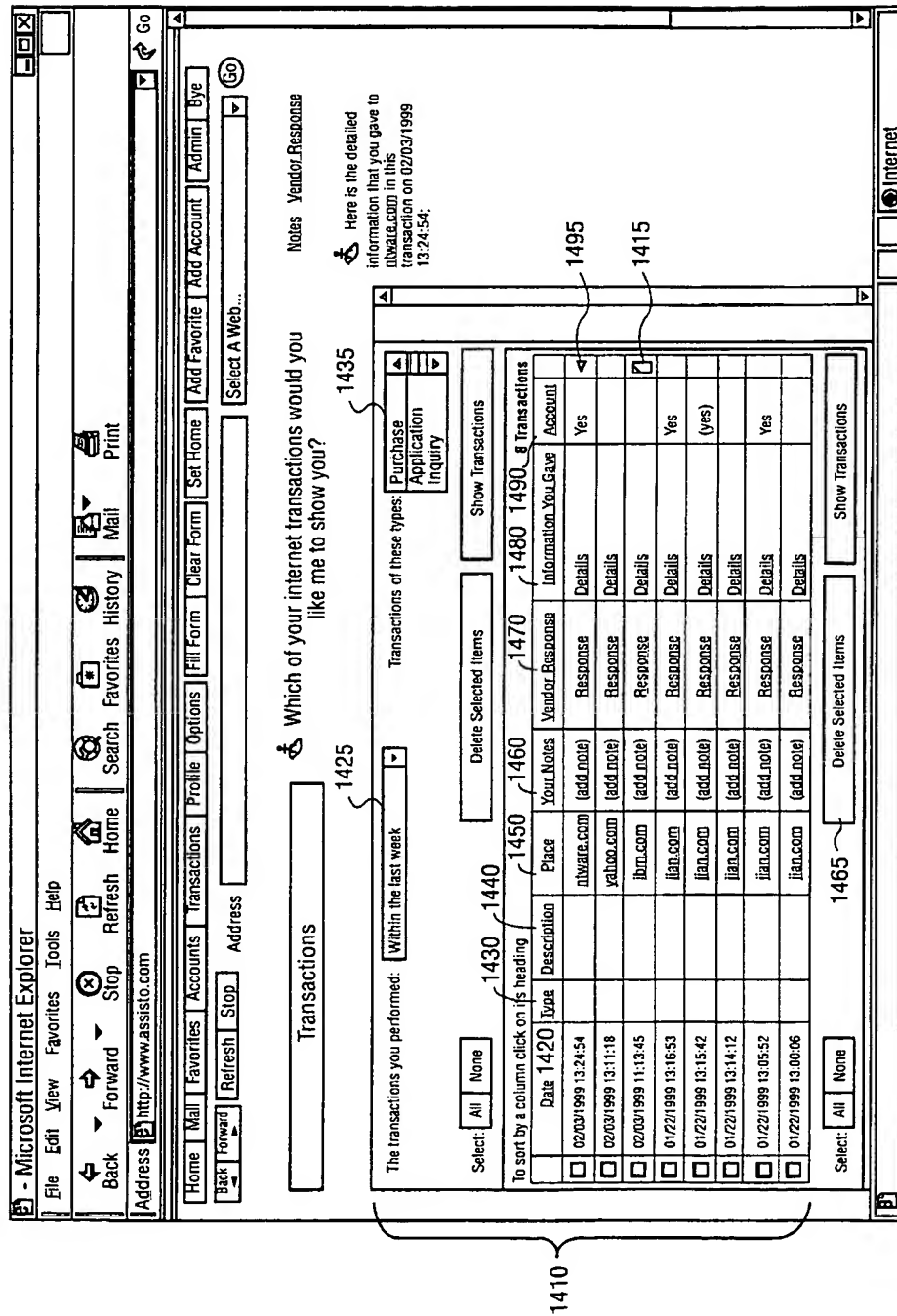
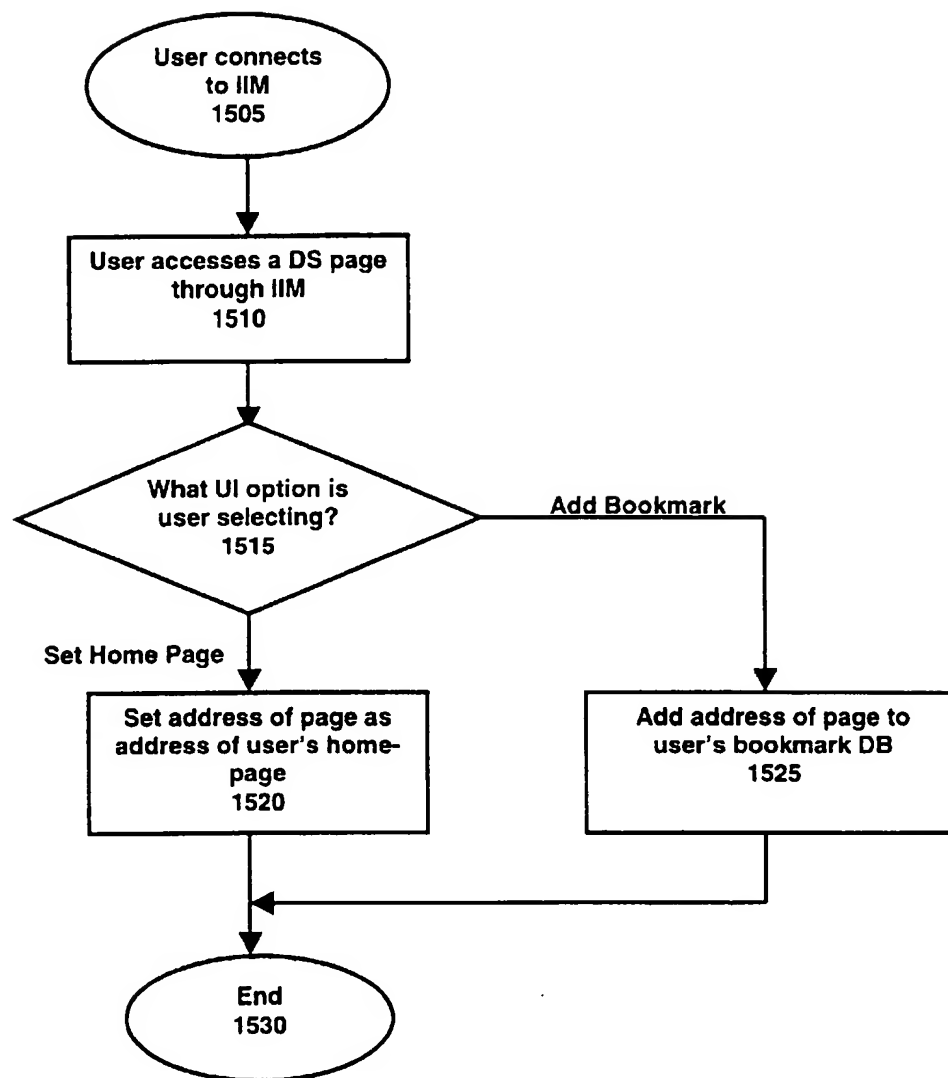
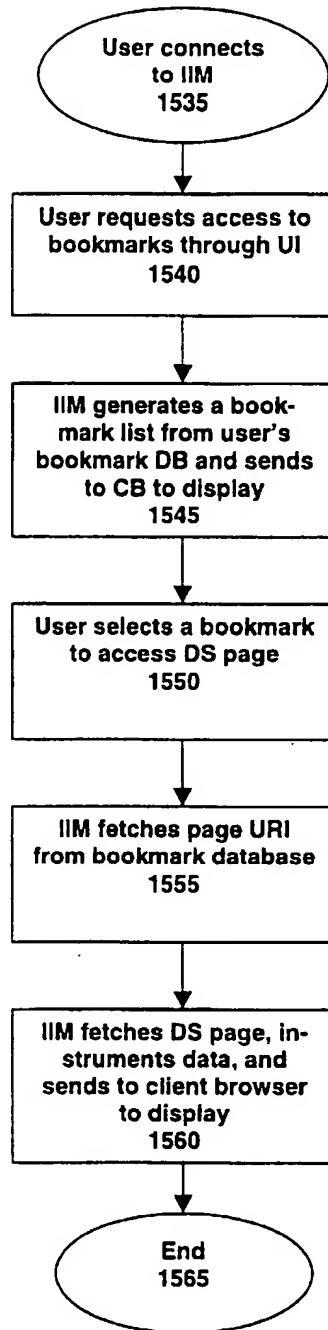


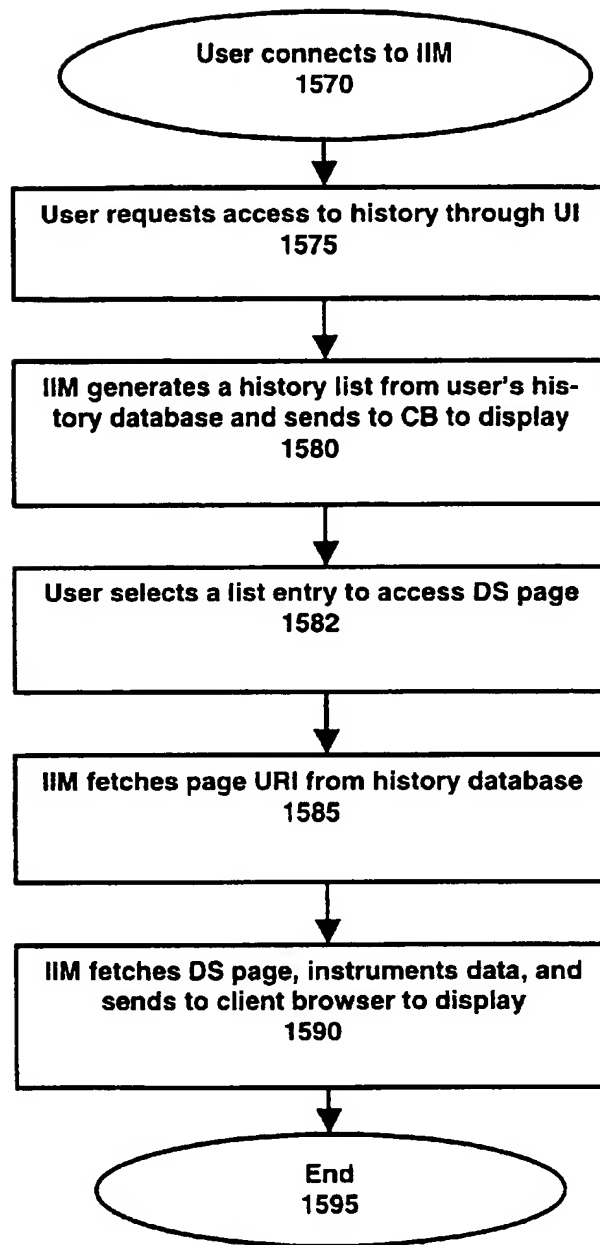
Fig. 13





**Fig. 15A**

**Fig. 15B**

**Fig. 15C**

Original Code	Altered Code	Comments
<b>HTML</b>		
<code>&lt;base href="anyURL"&gt;</code>	<code>&lt;base href="http://www.DS.com/myDocument.html"&gt;</code>  <code>&lt;base href="anyURL"&gt;</code>	www.DS.com is the hostname of the DS. The IIM inserts the first <BASE> tag line after the <HTML> tag and before the <HEAD> tag, and before any existing <BASE> tags
<code>&lt;form action="/actionURL"&gt;</code>	<code>SaveOrigAction( form, actionURL)</code>  <code>&lt;form action="http://www.IIM.com/redirect?act=http://www.DS.com/actionURL"&gt;</code>	www.IIM.com is the hostname of the IIM and www.DS.com is the hostname of the DS. SaveOrigAction() is a Javascript function that saves the form's original action.
<code>&lt;applet codebase="/codebase" code="applet.class"&gt;</code>	<code>&lt;applet codebase="http://www.IIM.com/redirect?cb=http://www.DS.com/codebase" code="applet.class"&gt;</code>	www.IIM.com is the hostname of the IIM and www.DS.com is the hostname of the DS.
<code>&lt;frame src="/myFrame.html"&gt;</code>  other tags, e.g., <script>, <area>, <layer>, <img>	<code>&lt;frame src="http://www.IIM.com/redirect?src=http://www.DS.com/myFrame.html"&gt;</code>	www.IIM.com is the hostname of the IIM and www.DS.com is the hostname of the DS.
<b>Javascript</b>		
<code>link.href = "newLocation"</code>	<code>setURLProperty ( link, "href", "newLocation" )</code>	setURLProperty() sets the value of the property href to the value "http://www.IIM.com/redirect?url=http://www.DS.com/newLocation", where www.IIM.com is the hostname of the IIM and www.DS.com is the hostname of the DS.
<code>link.onclick = originalOnClick</code>	<pre>function addNewLinkOnClick(link){ link.onclickOrig = link.onclick; link.onclick = newLinkOnClick; }  function newLinkOnClick(link){ if ( link.originalHref == null ) link.originalHref = link.href; var newHref = getFullPathName( link.originalHref ); link.href = http://www.IIM.com/redirect?url= newHref; return link.onclickOrig(); }</pre>	getFullPathName() returns the full pathname URL of the HTML and www.IIM.com is the hostname of the IIM. The function addNewLinkOnClick() is called when the HTML document is first loaded

Fig. 16A

Original Code	Altered Code	Comments
<code>document.write(StringToWrite)</code>	<code>writeDocument ( document, StringToWrite );</code>	<code>writeDocument()</code> recursively modifies all HTTP control points that occur in <code>StringToWrite</code>
<code>window.open( newLocation );</code>	<code>openWindow ( window, newLocation );</code>	<code>openWindow</code> calls <code>window.open()</code> with the argument "http://www.IIS.com/redirect?url= =http://www.DS.com/newLocati on"
<code>form.onsubmit = originalOnSubmit</code>	<pre>function addNewFormOnsubmit(form){ form.onsubmitOrig = form.onsubmit; form.onsubmit = newFormOnsubmit; }  function newFormOnsubmit ( form ) { if (form.originalAction == null) {form.originalAction = form.action;}  var newAction = getFullPathName(form.original Action);  form.action = http://www.IIM.com/redirect?url =newAction;  return form.onsubmitOrig(); }</pre>	<code>getFullPathName()</code> returns the full pathname URL of the HTML document and <code>www.IIM.com</code> is the hostname of the IIM. The function <code>addNewFormOnsubmit()</code> is called when the HTML document is first loaded
<b>Java:</b>		
<code>class java.net.Socket</code>	Extends <code>java.net.Socket</code> and overrides various constructors.	The extended method modifies the host and port arguments. The modified host argument is the hostname of the IIM. The modified port argument is the port of the IIM
<code>java.applet.AppletContext</code>	Extends <code>java.applet.AppletContext</code> and overrides various constructors	The extended method modifies the url argument. The modified url sends the HTTP request to the IIM with the full pathname of the original url as a query parameter
<code>class java.applet.Applet</code>	Extends <code>java.applet.Applet</code> and overrides various constructors	The extended method modifies the url argument. The modified url sends the HTTP request to the IIM with the full pathname of the original url as a query parameter

Fig. 16B

Original Code	Altered Code	Comments
<b>HTTP Headers</b>		
referer, e.g. value = origDoc.htm	referer, http://www.DS.com/origDoc.htm	referer value is replaced with the full pathname of the document's original URL
content-type, value = null	content-type, e.g., value = image/gif	If the value of content-type is null, the IIM sets this header to a value that describes the type of content contained in the document.
refresh, e.g. 5000; origDoc.htm	5000; http://www.IS.com/redirect?ref= http://www.DS.com/origDoc.htm	IIM replaces the URL portion of the value of the refresh header with the full pathname of the document's original URL
301, 302 status codes, e.g., URI value = http://www.DS.com/origDoc.html	301, 302 status codes, URI value = http://www.IS.com/redirect?url= http://www.DS.com/origDoc.html	www.IIM.com is the hostname of the IIM
201, 303, 305, 307 status codes URI values	201, 303, 305, 307 status codes modified URI values	See 301, 302 status codes

Fig. 16C

Original Code	Altered Code	Comments
<b>HTML</b>		
target = "_top"  for <a href>, <frame>, <form> and <base> tags	target = "DSFrameName"	where DSFrameName is the name of the DSDA top frame.
target = "_parent"  for <a href>, <frame>, <form> and <base> tags	target = "DSFrameName"	where DSFrameName is the name of the DSDA top frame, if the current window evaluates to the DSDA top frame.
<b>Javascript</b>		
top.locationProperty = value	setTopProperty ( currentWindow, locationProperty, value )	setTopProperty sets a location property on the DSDA top frame with name locationProperty to have value value
window.parent. locationProperty = value	setTopProperty ( window.parent, locationProperty, value )	setTopProperty sets a location property on the DSDA top frame with name locationProperty to have value value if window.parent evaluates to the top or IIM frame
<b>Java</b>		
java.applet.AppletContext.show Document( url, target )	newShowDocument ( window, url, target ) { java.applet.AppletContext.show Document( url, newTarget ) }	newShowDocument calls java.applet.AppletContext.show Document where newTarget is the name of the DSDA frame if target equals "_top" or if target equals "_parent" and window is the DSDA frame, "newTarget" is set to "target" otherwise.

Fig. 17

Original Code	Altered Code	Comments
<b>Javascript</b>		
string = document.cookie	string = getCookie( window, document )	getCookie() gets the cookie's value from the IIM and assigns the value to string
document.cookie = cookieString	setCookie( window, document, cookieString)	setCookie() sends the value of cookieString to the IIM to be managed
<b>HTTP Headers</b>		
	cookie	IIM sends this header to the DS on a need basis
<b>Java</b>		
javax.servlet.http.Cookie	<pre>public class ISCookie implements Serializable {      private long _creationTime;      public boolean hasExpired(){     // expiration function code } }</pre>	maintains the cookie's creation time and contains convenience routines that determine if the cookie has expired
Cookies	<pre>public class Cookies extends PersistentObject implements Serializable {      public String getCookie (URL url) { }      public static Cookie parseCookie(String str) { }      public void addCookie(ISCookie coo, URL url) { }      private boolean validCookie(ISCookie coo, URL url) { } }</pre>	extends the Java persistent object class, and saves in a user database the user-specific cookie information that the DS sent in the set-cookie header. This class also finds the cookies in the cookie database that are valid for a certain URI based on well known Cookie rules and returns a Cookie string for a given URI.

Fig. 18



Original Code	Altered Code	Comments
<b>Javascript</b>		
window.location = newLocation	setLocation(currentWindow, window, newLocation)	setLocation() sets the window.location to the value "http://www.IIM.com/redirect?url= http://www.DS.com/newLocation", where www.IIM.com is the hostname of the IIM and www.DS.com is the hostname of the DS.
saveLocation = window.location	saveLocation = getLocation(currentWindow, window)	where getLocation returns window.location except when window is equal to the top frame, then it returns the DSDA top frame's location
top.userProperty = value	setTopProperty ( currentWindow, userProperty, value )	where setTopProperty sets a user property on the DSDA top frame with name userProperty to have value value
window.parent.userProperty = value	setTopProperty ( window.parent, userProperty, value )	where setTopProperty sets a user property on the DSDA top frame with name userProperty to have value value if window.parent evaluates to the top frame.

Fig. 19

1

# CLIENT-SERVER INDEPENDENT INTERMEDIARY MECHANISM

## FIELD OF THE INVENTION

The present invention relates to client-server communication, and more specifically, to using an independent intermediary mechanism between a client and a server.

## BACKGROUND

The World-Wide Web (WWW, W3, the Web) is an Internet client-server hypertext distributed information retrieval system. An extensive user community has developed on the Web since its introduction.

FIG. 1 is a block diagram of a prior art client-server system. The client A 110 can access destination servers DS1-DS3 150-170. Similarly, other clients B and C, 120 130, can access the destination servers DS1-3 150-170. Each destination server may provide different services, information, or other data to the user.

On the Web everything (documents, menus, indices) is represented to the user as hypertext objects in hypertext markup language (HTML) format, or as Java, or JavaScript objects, or other data types. Hypertext links refer to other documents by their uniform resource identifiers (URIs). The client program, known as a browser, e.g. NCSA Mosaic, Netscape Navigator, runs on the user's computer and provides two basic navigation operations: to follow a link or to send a query to a server. Users access the web through these browsers.

Users often access the web from multiple locations. Users may access the web from their office, at different locations at work, at home, or on the road. Libraries and Internet cafes make web access available on a walk-in basis as well.

A user accesses a server by typing the URI of the server into the browser's address window. The browser then connects to the server corresponding to this URI. Another method of accessing a web site is by selecting the web site from list of bookmarks. The list of bookmarks is resident in the browser in the user's computer. Thus, if a user wishes to have similar bookmarks on multiple computers, he or she must manually copy the bookmarks and transfer them between the computers. This process is inconvenient.

Furthermore, many servers use cookies to store information about the user. This information may include the user name, password, previous interests, etc. These cookies are also stored in the user's browser. Again, this means that if the user is accessing the Internet from multiple computers, the user's cookies have to be duplicated into multiple computers. This process is inconvenient.

Many users have multiple accounts on different computer systems. For example, a user may have an account with a bank, an e-mail account, a personalized portal site account, and an account on an e-commerce server. Currently, the users must log into each of these accounts by remembering and providing his or her user name and password. For security, each of these user names and passwords should be different. Remembering different names and passwords is inconvenient to the user. Thus, a method for a simple log-in into various accounts from any computer would be advantageous.

Most clients and servers support "forms" which allow the user to enter arbitrary text as well as selecting options from customizable menus and on/off switches. As more business is transacted on the Web, forms are proliferating. The forms

2

may include forms for requesting further information, for ordering items from the Web, for registering for a Web site, etc. However, the user generally can not get a copy of the information filled into the form. The user can either print the page when the form is filled in, generating a paper copy, or rely on the server to respond in a manner that permits the user to make a record of the information entered in to the form. A method of tracking information filled into forms would be advantageous. Furthermore, vendors may respond with an order number or other useful information. The user can keep a copy of this page, which is generally only temporarily available, by printing it, or copying down the information provided. A method of attaching this vendor response to the original order information and making both available to the user would be advantageous.

Furthermore, currently, the user has to fill out each of these forms separately. Generally, the forms request the same types of information, i.e. name, address, telephone number, e-mail address, etc. The user has to enter all of this information for each form. This is repetitious and takes time. Additionally, if such information as credit card number or social security number is requested, the user has to pull out the credit card and copy a long string of numbers. This makes errors likely. Furthermore, the user has to verify that a Web site that requests a credit card number or similar confidential information is of the appropriate level of security for the user to feel comfortable sending the information over the Web. An improved method of filling out forms would be advantageous.

## SUMMARY OF THE INVENTION

A method and apparatus of a client-server independent intermediary mechanism (IIM) is described. The method comprises displaying a frame including a user interface of the IIM (IIM frame), and a second frame framing a destination server display area (DSDA). The method further comprises retrieving data for display from a destination server, and instrumenting the data prior to display such that future data retrieved from a destination server is displayed in the DSDA, without writing over the IIM frame.

## BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1 is a block diagram of a prior art client-server system.

FIG. 2 is a block diagram of one embodiment of the client-server system including the independent intermediary mechanism.

FIG. 3A is a block diagram of one embodiment of the client-server system including multiple independent intermediary mechanisms.

FIG. 3B is a block diagram of another embodiment of the client-server system including multiple independent intermediary mechanisms.

FIG. 4 is a block diagram of one embodiment of the independent intermediary mechanism.

FIG. 5 is a block diagram of one embodiment of the layout of the user interface of the independent intermediary mechanism.

FIG. 6 is a flowchart of an overview of using the independent intermediary mechanism.

FIG. 7 is a flowchart of one embodiment of the process of displaying information from a destination server through the independent intermediary mechanism.

3

FIG. 8 illustrates one embodiment of the user interface of the independent intermediary mechanism.

FIG. 9 illustrates another embodiment of the user interface of the independent intermediary mechanism.

FIG. 10 is a flowchart of one embodiment of the form fill functionality.

FIG. 11 is a flowchart of one embodiment of the learning process in the database.

FIG. 12A is a flowchart of one embodiment of adding accounts.

FIG. 12B is a flowchart of one embodiment of accessing an account through an auto-log-in feature.

FIG. 13 is a flowchart of one embodiment of the transaction management functionality.

FIG. 14 illustrates one embodiment of the listing of transactions.

FIG. 15A is a flowchart of one embodiment of selection of a home page or a bookmark.

FIG. 15B is a flowchart of one embodiment of using the bookmark functionality.

FIG. 15C is a flowchart of one embodiment of using the history functionality.

FIGS. 16A, 16B and 16C are tables illustrating examples of redirecting references to DS through IIM.

FIG. 17 is a table illustrating examples of making the IIM user interface frame persistent.

FIG. 18 is a table illustrating examples of accessing cookies from the

FIG. 19 is a table illustrating examples of preserving top frame or IIM frame integrity for DS.

#### DETAILED DESCRIPTION

A client-server independent intermediary mechanism is described. The independent intermediary mechanism (IIM) mediates information exchanged between a client and servers by having the client-server communication pass through the IIM. This allows the IIM to offer various services. For one embodiment, the IIM may be used to have a central web-accessible set of bookmarks. The IIM may further provide tracking of transactions on the web, providing a user-accessible transaction record. The IIM may further be used to fill in various forms automatically. The IIM may further be used to access multiple accounts, such as e-mail accounts, bank accounts, etc. with a single button. The IIM may further be used to store the user's profile, including passwords to various pages, etc. These and other uses of the IIM are described below.

FIG. 2 is a block diagram of one embodiment of the client-server system including the independent intermediary mechanism. Multiple clients A-C 210, 215, 220 access multiple destination servers (DSs) 280, 285, 290, through the independent intermediary mechanism (IIM) 250. Client A 210 is described as an example. It is to be understood that multiple clients are implemented in similar ways.

Client A 210 accesses the IIM 250. For one embodiment, this occurs when the user of the client A 210 accesses the web site hosting the IIM 250. When the IIM 250 is accessed, a new client component (CC) 230 is established. The client component(s) 230, 235, 240 and the server component 260 together form the IIM 250. For one embodiment, the IIM 250 is located on a server accessed by the client A 210 through an Internet connection. For another embodiment, the IIM 250 is located within the local Intranet of client A 210. For yet another embodiment, the IIM 250 is located on the client's own computer.

4

For one embodiment, the client component 230 is established on the local computer of the client 210. For another embodiment, the client component 230 is on a server, or on a third computer system. The client component 230 is created in response to the client 210 connecting to the IIM 250.

The client A 210 has a connection to the server component 260 through the client component 230. For one embodiment, the client A 210 also establishes a direct connection with the server component 260. This direct connection may be used to communicate certain information directly between the server component 260 and the client A 210. The client 210 accesses the destination servers DS1-3 280, 285, 290 through the IIM 250. For one embodiment, all of the communication between the destination server DS1 280 and the client A 210 is routed through the IIM 250. For another embodiment, certain communications are routed directly between the client A 210 and the destination server 280. For example, certain large images that do not invoke other images or other data may be routed directly in order to speed up processing.

The number of client components 230, 235, 240 depends on the number of clients 210, 215, 220 coupled to the server component 260 at any one time. For one embodiment, the server component 260 consists of multiple components that act together. A block diagram of one embodiment of the IIM 250 may be found in FIG. 4, below.

FIG. 3A is a block diagram of one embodiment of the client-server system including multiple independent intermediary mechanisms 350, 360. Each IIM 350, 360 is shown having a corresponding server component, 355, 365. For another embodiment, the server components 355, 365 may be located on a single server, or within a single IIM. Having server components 355, 365 coupled together may serve multiple purposes. For example, if a single IIM 350 has too many users connected to it, the IIM 350 may redirect users to a second IIM 360. For another embodiment, a user may log on to a local IIM 350, for speed reasons, and the local IIM 350 may connect to the user's "home" IIM 360 to retrieve the user's data. For yet another embodiment, the user can connect to their "home" IIM 350, which is remote, and the "home" IIM 350 may redirect the user to a local IIM 360 and send the user's data to the local IIM 360. In this way, the user's connection to the IIM 350, 360 may be optimized.

FIG. 3B is a block diagram of another embodiment of the client-server system including multiple independent intermediary mechanisms. In this example, a client 310 is coupled to two IIMs 350, 360. Generally, the client 310 first connects to the first IIM 350. Then, through the user interface of the first IIM, the client 310 connects to the second IIM 360. This may be advantageous if, for example, the first IIM 350 and second IIM 360 provide different services. Thus, for example, one IIM 360 may provide additional account management features, while the other IIM 350 provides form-fill features. By connecting to both IIMs 350, 360, in series, the user has access to the features provided by both IIMs 350, 360.

FIG. 4 is a block diagram of one embodiment of the independent intermediary mechanism. The IIM 400 has three layers. The lowest layer of the IIM 400 is the core engine 410. The core engine 410 includes a server component SC and a client component CC. The Server Component, for one embodiment, is resident on the server, and handles all remote actions. The Client Component, for one embodiment, is resident on the client's system, while the

5

client is connected to the IIM 400. For one embodiment, the client component is automatically removed from the client's system when the client disconnects from the IIM 400. The lowest layer also includes a Cookie Manager 413, for managing any cookies received from and being sent to the destination server. The use of such cookies is discussed in more detail below. Furthermore, the lowest layer may include a Activation Manager 416. The Activation Manager 416 determines if information is being transmitted by the destination server. For one embodiment, the Activation Manager 416 further determines if information is being initiated by a user's action. Information transmitted between the DS and the client is instrumented by the IIM 400, as will be described below. The Activation Manager 416 detects when the IIM 400 should review communication between the client and the DS.

The second layer is the application/UI framework layer 420. The application/UI framework layer 420 establishes the basic user interface and IIM engine. The application/UI framework layer 420 creates a persistent frame for the IIM 400. For one embodiment, the application/UI framework layer 420 further includes an instrumenting manager 425, for instrumenting data flowing from the destination server to the client, through the IIM 400. This process of instrumenting is described in more detail below.

The third layer is the applications layer. The applications layer includes multiple applications. The applications listed here are listed as an example, and are not a complete list. The applications layer, for example, may include a Navigation Manager 430. The Navigation Manager 430 permits a user to navigate from destination server to destination server using the IIM 400. The applications layer may further include a Transaction Manager 440.

The Transaction Manager 440 tracks the user's transactions, stores them, and makes them available for the user's review. Transactions are interactions in which a user submits information to a destination server, for example to order an item, ask a question, or otherwise interact with the destination server. The Transaction Manager 440 tracks the data submitted by the user, and any response from the destination server, and permits the user to access this information.

The Account Manager 450 permits the user to log into a variety of accounts, from e-mail to stock trading accounts, using a single click. The Account Manager 450 further permits the user to add accounts to this list. The Form Manager 460 permits the user to fill out forms encountered on destination servers via a single click. This is extremely useful for users that transact business on the web, and often fill out identical forms many times. The Profile Manager 470 is the database of the user's personal information. This information may be edited by the user, and is used to fill in forms via the form manager 460. The Database Manager 480 manages the various databases of the IIM 400.

The Bookmark Manager 490 permits the user to manage bookmarks maintained within the IIM 400. Having bookmarks, URIs of pages the user wishes to save, available in the IIM 400 permits the user to access his or her bookmark list from any computer.

The History Manager 495 permits the user to manipulate the history list of sites the user has previously visited. For one embodiment, the user can change the permanence of the history lists, for another embodiment, the user can delete certain sites from the history list.

The Homepage Manager 497 permits the user to set a homepage that is displayed when the user initially connects to the server providing the IIM 400.

6

As can be seen, the IIM provides multiple functionalities. A single IIM 400 may include all of the functionalities described above, additional functionalities, or some subset of these functionalities. The IIM's functionality may be extended with additional features.

FIG. 5 is a block diagram of one embodiment of the layout of the user interface of the independent intermediary mechanism. The client browser application window 510 is displayed by a browser, such as Netscape Navigator or Microsoft Internet Explorer. The client side display area (CSDA) 520 is the display area available in the browser application window 510. Most browsers have a toolbar and other displays within the browser application window 510. For one embodiment, the IIM is designed to minimize the area of the browser application window that is not the CSDA 520.

The CSDA 520 includes a toolbar frame 530. Although the tool bar frame 530 is referred to as a frame, for one embodiment, the tool bar frame 530 may be implemented in a non-frame format. For one embodiment, the tool bar may be implemented as a separate window. For another embodiment, the tool bar may be implemented as part of the display, not as a frame.

The CSDA 520 further includes a destination server display area (DSDA) 540. The DSDA 540 is the area in which all information from destination servers is presented.

The CSDA 520 further includes a communications frame 550. The communications frame 550 is for communication between the client side and server side of the IIM. Generally, the communications frame 550 is hidden from the user's view. Thus, the user would not see the communication between the client component and the server component.

FIG. 6 is a flowchart of an overview of using the independent intermediary mechanism. At block 610, the user connects to the IIM through the client browser. For one embodiment, this is done by typing the address of the IIM into the address window of the browser. For one embodiment, the IIM may be the preset homepage of the user, or a bookmark in the client browser.

At block 615, the user connects to a destination server (DS) through the IIM. For one embodiment, this is done by typing the address of the destination server into the address window of the IIM. For another embodiment, the user may select an address from a history list of previously visited sites, from a bookmark list in the IIM, or the destination server may be a preset homepage in the IIM. The IIM records the DS in the history database. The history database tracks the web sites that the user has visited in the past. Such a history database may be useful to permit backtracking, or to visit previously visited sites. For one embodiment, this history database is maintained for a fixed duration of time, or a user preset period of time. For another embodiment, the history database is maintained indefinitely.

At block 620, the process changes the reference to DS to go through the IIM and load the information from the DS in the DSDA, maintaining the IIM frame. This is described in more detail below.

At block 625, the IIM determines whether the user submitted information to the destination server. For one embodiment, the actual test is whether information that is "sensitive" or "of interest" is submitted to the DS. For example, if a user selected a radio button for the next display, the response would be "no" even though some information was submitted. For one embodiment, the answer to this query is yes only if information that is in the user's profile is submitted. For one embodiment, the answer to this

query is provided by the user through the user interface. If the answer is yes, the process continues to block 630.

At block 630, the user's communication with the DS is recorded in the user's transaction database. For example, if the user ordered an item from a destination server site, the form that was filled in by the user, including all of the information filled in, would be recorded in the transaction database. This transaction database is available to the user. The process then continues to block 635. If, at block 625, the answer was no, the process continues directly to block 635.

At block 635, the IIM forwards the communication, i.e. the information submitted by the user, to the DS. This communication includes relevant cookies. A cookie is a packet of information sent by a destination server to a browser and then sent back by the browser each time it accesses that server. Cookies can contain any arbitrary information the server chooses and they are used to maintain state between otherwise stateless transactions. Generally, cookies are maintained in a user's browser. However, for one embodiment, the IIM maintains the user's cookies. This permits a user to log into a site, and have the appropriate cookies available, no matter from what web client device or client browser the user accesses the site.

At block 640, the process determines whether the destination server responded to the user's submission of information. For one embodiment, some destination servers respond, with a thank you page, other data pertaining to order number, shipping code, delivery date, etc., when information is submitted to them. If the destination server responds at block 640, the process continues to block 645.

At block 645, the DS's response is recorded in the user's transaction database, and associated with the user's submitted information. Thus, when the user reviews the transaction, he or she can review the entire transaction, including the DS's response.

At block 650, the IIM instruments the DS's response, stores any cookies returned by the DS, and forwards the response to the client browser. One embodiment of this process is illustrated in more detail in FIG. 7, below. Tables of some results of the process of instrumenting are illustrated in FIGS. 16A-C, and FIGS. 17-19.

At block 655, the process tests whether the user continues to browse through the IIM. The user continues to browse, the process returns to block 615. Otherwise, the process ends at block 660.

FIG. 7 is a flowchart of one embodiment of the process of instrumenting data displayed from a destination server through the independent intermediary mechanism. For one embodiment, FIG. 7 is a more detailed flowchart of block 650, in FIG. 6. At block 705, the IIM receives a communication from the DS. For one embodiment, this occurs in response to a user contacting a DS through the IIM.

At block 710, the process tests whether there is a cookie or multiple cookies associated with the communication. Cookies may be sent by the DS to the client, to be stored on the client browser. If a cookie is associated with the communication, the process continues to block 715. At block 715, the IIM cookie database is updated with the new cookie. For one embodiment, cookies sent by the DS to the client browser are handled through the IIM. Thus, the IIM would store all of the cookies for a DS, and give the DS its cookies. This is advantageous because it permits a user to access a DS from any computer, and all of the user's cookies are immediately available through the IIM. The process then continues to block 725. If no cookies were associated with the communication, the process continues directly to block 725.

At block 725, the process parses the code to find the next keyword. For one embodiment, keywords are tags in HTML, or known keywords in Java or JavaScript. FIGS. 16-19 illustrate some examples of keywords that may trigger this process. For another embodiment, keywords may be any triggering signal that indicates that an action may be performed.

At block 730, the process tests whether a keyword was found. If no keyword was found, the process continues to block 735, and ends. If the communication has no remaining keywords, the document has been fully instrumented, and is ready for display to the user. For one embodiment, certain communications may have no keywords at all. In that case, this process would end after a single pass. For yet another embodiment, under some circumstances, the process may ignore certain keywords. Certain references are not altered in the communication. For example, references that call static images, images that do not communicate information to the user and do not have embedded references, may be of no interest. For example, if the keyword calls a large passive figure with multiple components, the process may ignore the entire figure, by tagging figure related communications, and exit out of this process even if keywords remain. By altering only those references that are of interest, the process may be sped up. If a keyword was found, the process continues to block 740.

At block 740, the process tests whether the keyword is an attempt to access a cookie from the cookie database. If the keyword is an attempt to access a cookie, the process continues to block 745. At block 745, the access attempt is changed to fetch the cookie from the IIM's cookie database. Some examples of this process are provided in FIG. 18. For one embodiment, the IIM's cookie database may access the client browser's cookie database in order to determine whether there are additional cookies on the client browser. For one embodiment, the IIM can, with the user's permission, copy cookies from the browser cookie database to the IIM. This simplifies moving from direct access of a DS to accessing a DS through the IIM. The process then continues to block 750.

If the keyword is not an attempt to access a cookie, the process continues directly to block 750.

At block 750, the process tests whether the keyword is an attempt to access the top frame or IIM frame. If the keyword is an attempt to access the top frame or IIM frame, the process continues to block 755. At block 755, the access attempt is changed to access the top area of the destination server display area (DSDA). Some examples of this process are provided in FIG. 17. The process then continues to block 760.

If the keyword is not an attempt to access the top of IIM frame, the process continues directly to block 760.

At block 760, the process tests whether the keyword is a reference to the destination server. If the keyword is a reference to the destination server, the process continues to block 765. At block 765, the reference is changed to be fetched through the IIM. Some examples of this process are provided in FIGS. 16A-C. The process then continues to block 770.

If the keyword is not a reference to the destination server, the process continues directly to block 770.

At block 770, the process tests whether the keyword is an attempt to access data from the top frame or IIM frame. If the keyword is an attempt to access data from the top frame or IIM frame, the process continues to block 775. At block 775, the access attempt is changed to fetch data from the

topmost frame of the DSDA. Some examples of this process are provided in FIG. 19. The process then returns to block 725, and parses to find the next keyword.

For one embodiment, the above process may be triggered by a user. For example, a user may select a link, activate a JavaScript function, or otherwise initiate communication between the destination server and the client. The same process may occur in response to a cookie being sent or received, or a keyword being found as described above with respect to FIG. 7.

FIG. 8 illustrates one embodiment of the user interface of the independent intermediary mechanism. The user interface includes a browser toolbar 805. For one embodiment, the IIM may configure the browser such that the browser toolbar area 805 is not displayed when the IIM is active. The display area 810 of the browser includes the IIM toolbar 820, a hidden communications frame 815, and the destination server display area 845.

The IIM toolbar 820 includes the known browser controls 825, such as back, forward, refresh, stop, etc. Additional browser controls 825 may be added. The toolbar 820 further includes an address entry control 830, where a user can type a destination server address in order to access the DS.

The IIM toolbar 820 may further include buttons, or other selection mechanisms that permit a user to configure and use the IIM. The buttons may include Home, selecting a user's preset homepage, etc. The homepage is preset using the Set Home button 852. The buttons may further include the Mall button, giving one-button access to shopping. The buttons may further include Tags 860, displaying a list of a user's bookmarks. Bookmarks are added by selecting the Tag Address while visiting a web site, or by selecting the Tag Address button 862, and typing the address of a location to be bookmarked.

The buttons may further include Accounts 865, permitting single-button log-on to a variety of accounts. These accounts are added with the Add Account button 867, as will be described below.

The buttons may also include a Transactions button 870, that permits a user to review his or her transactions. This is illustrated in the destination server display area 845 of FIG. 8. The Profile button 875 permits the user to enter his or her personal data. The Fill-Form button 880 permits the user to fill in a form using the personal data from the user's profile or by using information submitted previously using the same form. If a form is displayed on the destination server display area 845, and the user selects the fill-form button 880, the form is automatically filled in with the user's information. The Clear Form button 882 permits a user to remove the information filled into a form. This provides an additional level of security to the user.

The Admin button 885 provides access to account administration services. For one embodiment, the Admin button 885 is only available to those users who are authorized administrators. For one embodiment, the Admin button 885 is only displayed if the user is authorized to access account administration services.

The toolbar 820 further includes a Bye button 890, which logs off the user from the IIM. The toolbar 820 illustrated is exemplary. The content and organization of the buttons on the toolbar 820 may be changed without changing the invention.

FIG. 9 illustrates another embodiment of the user interface of the independent intermediary mechanism. As can be seen, the user interface may be flexibly implemented. Certain features may be provided by one interface and not

provided by another. Furthermore, the look and feel of the user interface may be altered. The user may, for example, access all of the IIM features through pull-down menus, such as the pull-down menu 935, or radio buttons instead of buttons. One skilled in the art understands other types of user interface changes that may be made without departing from the broader spirit and scope of the invention as set forth in the appended claims.

FIG. 10 is a flowchart of one embodiment of the form fill functionality. At block 1010, a document with a form is displayed. For one embodiment, this is a result of a user accessing a destination server location that includes a form. This form may be an order form, an information request form, or any other form that may be encountered on the Web.

At block 1015, the user requests the form-fill function through the IIM user interface. For one embodiment, the user presses the form-fill button. For another embodiment, the form fill may be automated. For yet another embodiment, the user can select whether the form fill function is automatically engaged.

At block 1020, the process determines whether the form is in the user's transaction database. The user's transaction database has records of previously accessed and filled-in forms for the particular user. The transaction database may maintain such records for a limited time, or the user may delete transaction records. Thus, merely because a user has been to a particular site previously may not mean that the form is in the user's transaction database. If the form is in the user's transaction database, the process continues to block 1040, otherwise, the process continues to block 1025.

At block 1025, the process determines whether the form is in the form database. The form database is maintained by the IIM and includes "known" forms. Such known forms have associations between form control identifiers in the form and profile items. Thus, for example, a form control identifier that is labeled "name" may have a link to the "First Name Last Name" item in the user profile. If the form is known, the process continues to block 1030. At block 1030, the form control identifiers in the form are filled in from the user profile. The process then returns to block 1055.

If the form is not known, the process continues to block 1035. At block 1035, the form controls are identified, based on the name of each control. Each control name is associated with entries in the user profile. The process then continues to block 1030, and the data is filled into the form from the user profile. For one embodiment, block 1035 is skipped. This type of "guessing" may be user enabled, or may be only attempted for forms that are similar to known forms.

At block 1020, if the form was found in the user's transaction database, the process continued to block 1040.

At block 1040, the process tests whether any data in the user profile has been changed since the transaction in the transaction database was recorded. Transaction records are dated, as are changes to the user profile. A user profile may be changed by the user, for example, to change a credit card expiration date, number, or home address. If a user profile change of a relevant field is dated after the transaction record date, the process continues to block 1045, otherwise, the process continues directly to block 1050.

At block 1045, the changed information is filled in from the user profile. In this way, the user only had to update his or her records once, in the profile, and that change is carried through the IIM. For one embodiment, this step may be skipped. For another embodiment, this step may be user enabled.

At block 1050, the remaining form control identifiers in the form are filled with data from the transaction database. The process then continues to block 1055.

## 11

At block 1055, the filled-in form is displayed to the user, and the user is permitted to edit the data in the form. The user, for example, may not wish to provide certain data to a destination server. The user may chose to erase such data. Alternatively, the form may request data that is not found in the user's profile. The user may chose to fill in such data.

At block 1060, the user submits the form to the destination server. For one embodiment, the IIM stores the information submitted to the server in the user's transaction database. This is illustrated in FIG. 13 below. At block 1065, the process ends. For one embodiment, the user may optionally select whether to use the user profile, transaction database, or both, and in what order, for form fill functions.

FIG. 11 is a flowchart of one embodiment of the learning process in the database. At block 1110, a privileged user connects to the IIM. For one embodiment, this privileged user is an employee of the group maintaining the IIM. For another embodiment, this "user" is an artificial intelligence unit that is used to identify forms, as will be described below. Such intelligent recognition programs are known in the art.

At block 1120, the privileged user accesses a destination server page with a form through the IIM. At block 1130, the IIM displays a user interface for cataloguing the form.

At block 1140, the user maps each form control to an element in the user profile object. the user profile is set up to contain a large number of possible data elements. Each form control should have a corresponding profile element. If no profile element is found for a form control, that form control may be tagged as "form specific." For one embodiment, multiple elements in the user profile may be associated with a single form control, or vice versa.

At block 1150, other information about the form is added. This information may include such information as the address of the form, whether the connection with the destination server that serves the form is a secure connection, whether the form is of a particular classification, etc.

At block 1160, the user submits the information to the IIM.

At block 1170, the IIM updates the form identification and form description in the form database to include the information added by the user. For one embodiment, the updating is a periodic batch updating. For one embodiment, a single central form database is maintained. In that instance, the IIM's updating may include sending the new form to other IIMs. Alternatively, each IIM may maintain its own separate form database. For yet another embodiment, an IIM may have a central form database, and a separate internal form database. This may be useful, for example, for an IIM implemented within a company which has the general form database for pages accessed outside the company, and a separate internal database for internal web page forms.

At block 1180, the process ends. Of course, the privileged user may enter multiple entries, and may start the process again at block 1120.

FIG. 12A is a flowchart of one embodiment of adding accounts. At block 1210, the user connects to the IIM through a client browser. At block 1220, the user accesses a destination server through the IIM. For one embodiment, the user accesses the account log-in page of the DS. This may be, for example, the account log-in page of the user's bank, of a portal, or of any other DS.

At block 1230, the user requests to add the account to the user's account database. Each user may have an account database, which includes a list of accounts the user can access with a single click.

## 12

At block 1235, the process determines whether the user has submitted login information to the account log-in page. If the user has not submitted the information, the process continues to block 1240, and the user is prompted to complete the log-in process. For one embodiment, if the account log-in process includes multiple pages, the user may indicate the end of the log-in process by pressing a certain key, or through other means. The process then continues to block 1245. If the user has submitted all of the log-in information, the process continues to block 1245 directly.

At block 1245, the account entry is added to the user's account database. The account log-in information and data of account entry creation are recorded. For one embodiment, further information may be recorded. For yet another embodiment, only the user's log-in procedure is recorded.

At block 1250, the account information is submitted to the DS for login. At block 1255, the process ends.

FIG. 12B is a flowchart of one embodiment of accessing an account through an auto-log-in feature. At block 1260, the user connects to the IIM. At block 1265, the user accesses the account auto-log-in feature using the IIM user interface. For one embodiment, this is done by the user pushing the account button.

At block 1270, the user selects an account to log into. For one embodiment, the user may have multiple accounts. In that instance, the IIM displays the accounts that the user has. For another embodiment, if the user only has a single account, that account is automatically selected when the user accesses the auto-log-in feature.

At block 1275, the IIM retrieves login information from the user's account database. As discussed above, the user's previous account log-in is monitored and recorded. This information is retrieved at block 1275.

At block 1280, the IIM sends the log-in information to the appropriate destination server to log-in the user. The account information includes the address of the DS. The IIM accesses the DS as a client, and sends the user's information.

At block 1285, the IIM instruments the DS's response and sends it to the user's browser for display. As discussed above, the response is instrumented such that references of interest are routed through the UM. The user can now use the account, as usual. At block 1290, the process ends.

FIG. 13 is a flowchart of one embodiment of the transaction management functionality. At block 1310, the user connects to the IIM.

At block 1320, the user transmits information in a form to the destination server. For one embodiment, the user first accesses a destination server page including a form through the IIM. This form may be an order form, an e-mail form, or any other type of form; The user then fills in the form and submits it to the DS. For one embodiment, the user may use the form-fill method described above to fill-in the form.

At block 1330, the process determines whether the user sent the user's e-mail address to the DS. The user may submit his or her e-mail address so the DS can send responses directly to the user's e-mail. For example, certain systems may send confirmation e-mails or alert notices to the user via e-mail. If the user submitted his or her e-mail address, the process continues to block 1340. Otherwise, the process continues directly to block 1350.

At block 1340, the e-mail address submitted to the DS is altered. Specifically, the e-mail address is bifurcated, generated two e-mails. The first e-mail goes to the user's e-mail address, as entered. The second e-mail goes to the IIM. The second e-mail includes in its address the IIM and the



## 13

transaction tag that identifies the transaction number to which the e-mail belongs. This allows the IIM to handle the e-mail. The process then returns to block 1350.

At block 1350, the IIM records a transaction in the user's transaction database and associates the submitted information with the transaction. The transaction, for one embodiment, has a transaction number.

At block 1360, the IIM determines whether there is a response from the DS. If there is a response, the process continues to block 1370. Otherwise, the process continues directly to block 1380.

At block 1370, the IIM records the response from the DS in the user's transaction database. For one embodiment, the destination server may respond to the user. This response is associated with the transaction record. In this way, the user may review the transaction record, including the response.

At block 1380, further information is recorded about the transaction. For one embodiment, this information may include the date and time of the transaction, and other information.

At block 1390, any notes, data, or e-mails received with the transaction tag are attached to the transaction. This may occur at any time, while the transaction is being recorded, or after that. The user may attach any data to the transaction, and the IIM may automatically attach any e-mails received with the transaction tag.

At block 1395, the process ends.

FIG. 14 illustrates one embodiment of the listing of transactions. The transaction list 1410 may be sorted by date, using a menu 1425. The transactions may also be sorted by type 1435. For one embodiment, alternative methods of searching transactions may also be implemented. For example, a user may search the transaction records for purchases from a certain destination server.

Each transaction record may include one or more of the following: date 1420, transaction type 1430, and description 1440 of the transaction. The record may further include the place 1450, the location from where the transaction was recorded. The user may add and edit additional notes 1460. Furthermore, the user may also add attachments 1415 to the transaction record. For example, the user may attach e-mails, documents, video, or other types of data. For one embodiment, e-mails may be redirected through the IIM and automatically attached to the transaction.

The vendor response 1470 is also recorded. The information the user provided 1480 during the transaction is also included in the transaction record. The transaction may further include the information whether the transaction belongs to one of the accounts 1490 in the user's account database. The user is permitted to delete selected transaction records using a delete button 1465.

FIG. 15A is a flowchart of one embodiment of selection of a home page. The user connects to the IIM at block 1505.

At block 1510, the user accesses a destination server page through the IIM. At block 1515, the process determines which option the user is selecting.

If the user is selecting the add bookmark option, the process continues to block 1525. At block 1525, the address of the page is added to the user's bookmark database. This database is accessible to the user, to permit the user to access various web sites without typing the address of the site. The process then continues to block 1530, and ends.

If the user selected the set home page option at block 1515, the address of the page is set as the user's homepage. The user's homepage is called up when the user initially

## 14

connects to the IIM. For one embodiment, the homepage is preset. For another embodiment, the user may not alter the homepage, and the homepage is customizable but includes advertising. The process then continues to block 1530, and ends.

FIG. 15B is a flowchart of one embodiment of using the bookmark functionality. At block 1535, the user connects to the IIM. At block 1540, the user requests access to the user's bookmarks through the IIM user interface. For one embodiment, the user requests the bookmarks by pressing the "Tags" button on the user interface.

At block 1545, the IIM generates a bookmark list from the user's bookmark database, and sends the list to the client browser to display. For one embodiment, the bookmark list is displayed in the destination server display area. For another embodiment, the bookmark list is displayed in a separate window, or a separate frame.

At block 1550, the user selects a bookmark to access a destination server page.

At block 1555, the IIM fetches the page address corresponding to the selected bookmark from the bookmark database. The bookmark database includes the actual address of the bookmark.

At block 1560, the destination server page is fetched by the IIM. The data from the destination server is instrumented and is sent to the client browser for display. In this way, the user can access bookmarks stored in the IIM's bookmark database. The process then continues to block 1565, and ends.

FIG. 15C is a flowchart of one embodiment of using the history functionality. At block 1570, the user connects to the IIM.

At block 1575, the user requests access to the history list through the IIM user interface. The history list includes the sites the user previously visited. For one embodiment, the history list is maintained for only a period of time, such as thirty days. For another embodiment, the history list is maintained indefinitely, and may be purged by the user.

At block 1580, the IIM generates a history list from the user's history database, and sends the history list to the client browser for display. For one embodiment, the history list is displayed in the destination server display area. For another embodiment, the history list is displayed in a separate window, or a separate frame.

At block 1582, the user selects a list entry to access the destination server page. At block 1585, the IIM fetches the page address from the history database. The page address is referenced through the IIM.

At block 1590, the IIM fetches the destination server page, instruments the communication, and sends the data to the client browser for display. At block 1595, the process ends. In this way, the IIM permits a user to access a variety of services through the IIM.

FIGS. 16A-C show sample alterations of references from the destination server by the IIM. FIGS. 16A-C illustrate changes to HTML, HTTP protocol, JavaScript, and Java. For one embodiment, this technique may be expanded to new languages and other types of interfaces. The data that is normally communicated directly between a Destination Server (DS) and client browser is altered by the IIM, as shown by FIGS. 16A-C. For one embodiment, some data may be transmitted directly between the DS and the client browser, without passing through the IIM.

For one embodiment, the IIM performs a subset of the message modifications required for redirection and down-



15

loads the client component to the client's browser, which performs the remaining subset of message modifications on the client machine. Together these two subsets of message modifications provide a complete solution for using an independent intermediary mechanism between a client and a server.

The modification of HTTP communication messages for redirection occurs on both the IIM and the client browser using the client component. The points at which the message modifications occur are called "HTTP control points".

FIGS. 16A-C illustrate examples of HTTP control points that occur on the client browser and the IIM. For HTTP message documents, description of modification code covers the three programming languages that are most widely used today for HTTP communication: HTML, JavaScript and Java. For another embodiment, the IIM utility may be broadened to include HTTP control points in other programming languages used for HTTP message documents. For one embodiment, the protocol modified in the messages is defined by the HTTP specification standard. One skilled in the art would understand how to expand the technique described to different programming languages or message protocols.

FIG. 17 is a table illustrating examples of making the IIM user interface frame persistent. The IIM prevents DS's from overwriting the user interface of the IIM. This permits the user to access the IIM regardless of what DS he or she is accessing.

FIG. 18 is a table illustrating examples of accessing cookies from the IIM. Generally, the destination server and destination server data on the client system access the cookie cache on the client's computer system. The IIM modifies the access mechanisms to access cookies from the IIMs cookie database.

FIG. 19 is a table illustrating examples of preserving top frame or IIM frame integrity for DS. Objects are often hung from the top frame of the client browser. The IIM changes the references to the top frame to create or access these objects to references to the top frame of DSDA. In this way, the objects are appropriately handled.

FIGS. 16-19 list some sample alterations resulting from the code instrumenting described above. Alternative methods of altering the code may be used. One skilled in the art knows how to implement different changes.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method of accessing data through an independent intermediary mechanism (IIM), the method comprising:
  - displaying a frame including a user interface of the IIM, the frame framing a destination server display area (DSDA);
  - retrieving destination server data (DS data) for display from a destination server;
  - instrumenting the DS data prior to display such that future data retrieved from the destination server is displayed in the DSDA, without writing over the frame displaying the user interface of the IIM.
2. The method of claim 1, wherein the step of instrumenting data prior to display comprises replacing a reference to a top frame or IIM frame with a reference to a top of the DSDA.

16

3. The method of claim 2, wherein said step of replacing comprises,
 

- in HTML, determining if a value of a Target attribute is "\_top", and changing the value to represent a topmost area of the DSDA.

4. The method of claim 2, wherein said step of replacing comprises,
 

- in Java, determining if a value of a Target attribute is "\_top", and changing the value to represent a topmost area of the DSDA.

5. The method of claim 2, wherein said step of replacing comprises, in JavaScript, replacing the reference to "top" with a reference to a topmost area of the DSDA.

6. The method of claim 1, further comprising:

altering requests for cookies such that cookies relevant to the destination server are accessed from the IIM.

7. The method of claim 6, wherein cookies received from the destination server or created by the DS data are stored in a user's portion of the IIM.

8. The method of claim 1, further comprising:

determining if a user's portion of the IIM includes a cookie for the destination server, and serving the cookie to the destination server and to the DS data, if the user's portion includes the cookie.

9. The method of claim 8, further comprising determining if a browser includes the cookie, and if the browser includes the cookie:

serving the cookie to the destination server and the DS data; and

saving the cookie in the user's portion of the IIM.

10. The method of claim 1, wherein at least one reference in the DS data to other DS data is redirected through the IIM.

11. The method of claim 10, wherein for predefined JavaScript, HTML and other code, the step of instrumenting is performed on a server side of the IIM, and wherein for dynamically generated code, the step of instrumenting is performed on a client side of the IIM.

12. The method of claim 1, wherein the step of altering data prior to display comprises replacing the DS data references to a reference through the IIM.

13. The method of claim 12, wherein only selected references are routed through the IIM.

14. The method of claim 12, wherein said step of replacing comprises altering a language of the reference such that any parameter which when set causes a document to be fetched from the destination server causes the document to be fetched through the IIM.

15. The method of claim 1, wherein links and references invoked by a user's selection are altered when the user selects the reference.

16. An independent intermediary mechanism (IIM) comprising:

a core engine retrieving destination server data (DS data) for display from a destination server;

a user interface framework for maintaining a frame including the IIM user interface on a client browser as the client browser accesses different destination servers.

17. The IIM of claim 16, further comprising:

a cookie database;

a cookie modification engine that alters a request for a cookie from the destination server or the DS data, such that the cookie relevant to the destination server is accessed from the IIM cookie database; and

the cookie modification engine further for maintaining and updating the cookie.

## 17

18. The IIM of claim 16, further comprising:

a data modification engine for instrumenting the DS data such that future data retrieved from the destination server is retrieved through the IIM.

19. A method of accessing data through an independent intermediary mechanism (IIM), the method comprising:

retrieving destination server data (DS data) for display from a destination server;

instrumenting the DS data such that future data retrieved from the destination server is retrieved through the IIM.

20. A method of accessing data through an independent intermediary mechanism (IIM), the method comprising:

retrieving destination server data (DS data) for display from a destination server;

## 18

altering a request for a cookie from the destination server or the DS data, such that the cookie relevant to the destination server is accessed from the IIM; and storing and updating the cookie in the IIM cookie database.

21. A communications mechanism comprising:

a first independent intermediary mechanism (IIM) displaying a frame including a user interface of the IIM, the frame framing a destination server display area (DSDA);

the first IIM retrieving destination server data (DS data) for display from a destination server and instrumenting the DS data prior to, the first IIM further for providing services to the user.

\* \* \* \* \*



US006728884B1

(12) **United States Patent**  
**Lim**

(10) Patent No.: US 6,728,884 B1  
(45) Date of Patent: Apr. 27, 2004

(54) **INTEGRATING HETEROGENEOUS AUTHENTICATION AND AUTHORIZATION MECHANISMS INTO AN APPLICATION ACCESS CONTROL SYSTEM**

6,412,070 B1 \* 6/2002 Van Dyke et al. .... 713/200

\* cited by examiner

(75) Inventor: **Mario Lim, South San Francisco, CA**  
(US)

*Primary Examiner*—Gilberto Barron

**Assistant Examiner—Courtney D. Fields**

(74) *Attorney, Agent, or Firm*—Marcel K. Bingham;  
Hickman Palermo Truong & Becker LLP

(73) Assignee: Entrust, Inc., Plano, TX (US)

(57) **ABSTRACT**

(\*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

A method and apparatus are provided for selectively authenticating and authorizing a client seeking access to one or more protected computer systems over a network. A request of a client to access one of the computer systems is received. A proxy security server is requested to authenticate the client using information identifying the client. An authorization of the client from the proxy security server is received, based on authentication results received from a remote security server that is coupled to the proxy security server. In response, access rights of the client are established, based on one or more access information records received from remote security server through the proxy security server. As a result, one or more legacy security servers may be easily integrated into an application access system without complicated modifications to the application access system.

(21) Appl. No.: 09/410,509

(22) Filed: Oct. 1, 1999

(51) **Int. Cl.**<sup>7</sup> ..... **H04L 9/32; H04L 12/22**

(52) U.S. Cl. .... 713/201; 709/219; 709/225;  
709/229; 713/155

(58) **Field of Search** ..... 713/182; 709/225,  
709/229, 219

(56) **References Cited**

## U.S. PATENT DOCUMENTS

6.182.142 B1 \* 1/2001 Win et al. .... 709/229

**29 Claims, 8 Drawing Sheets**

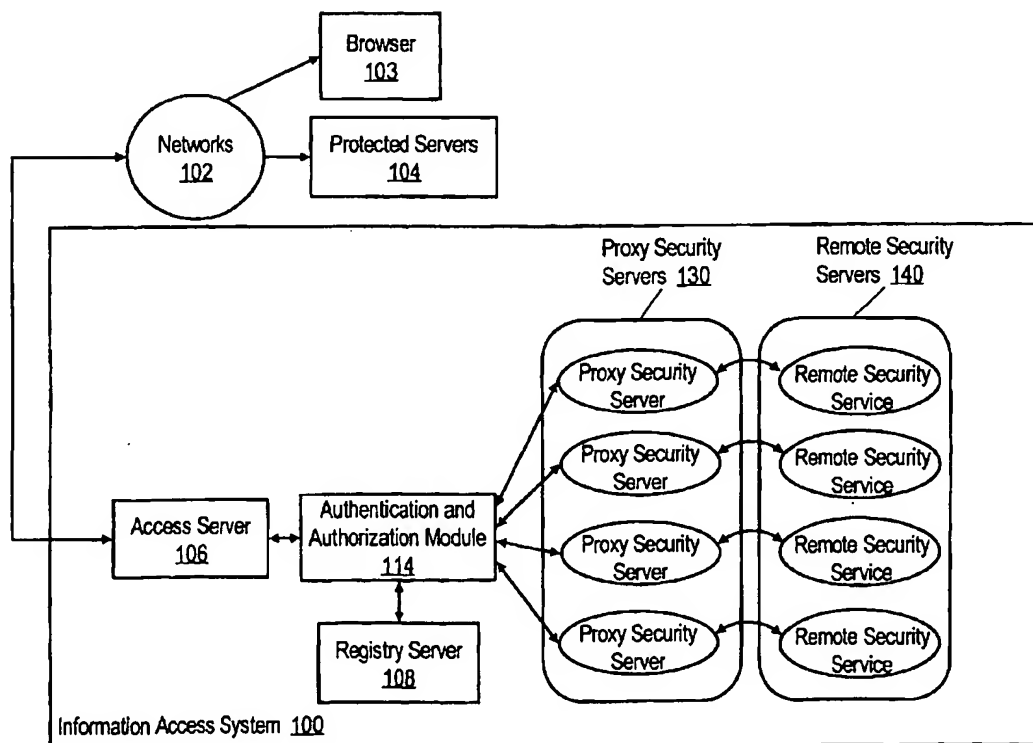
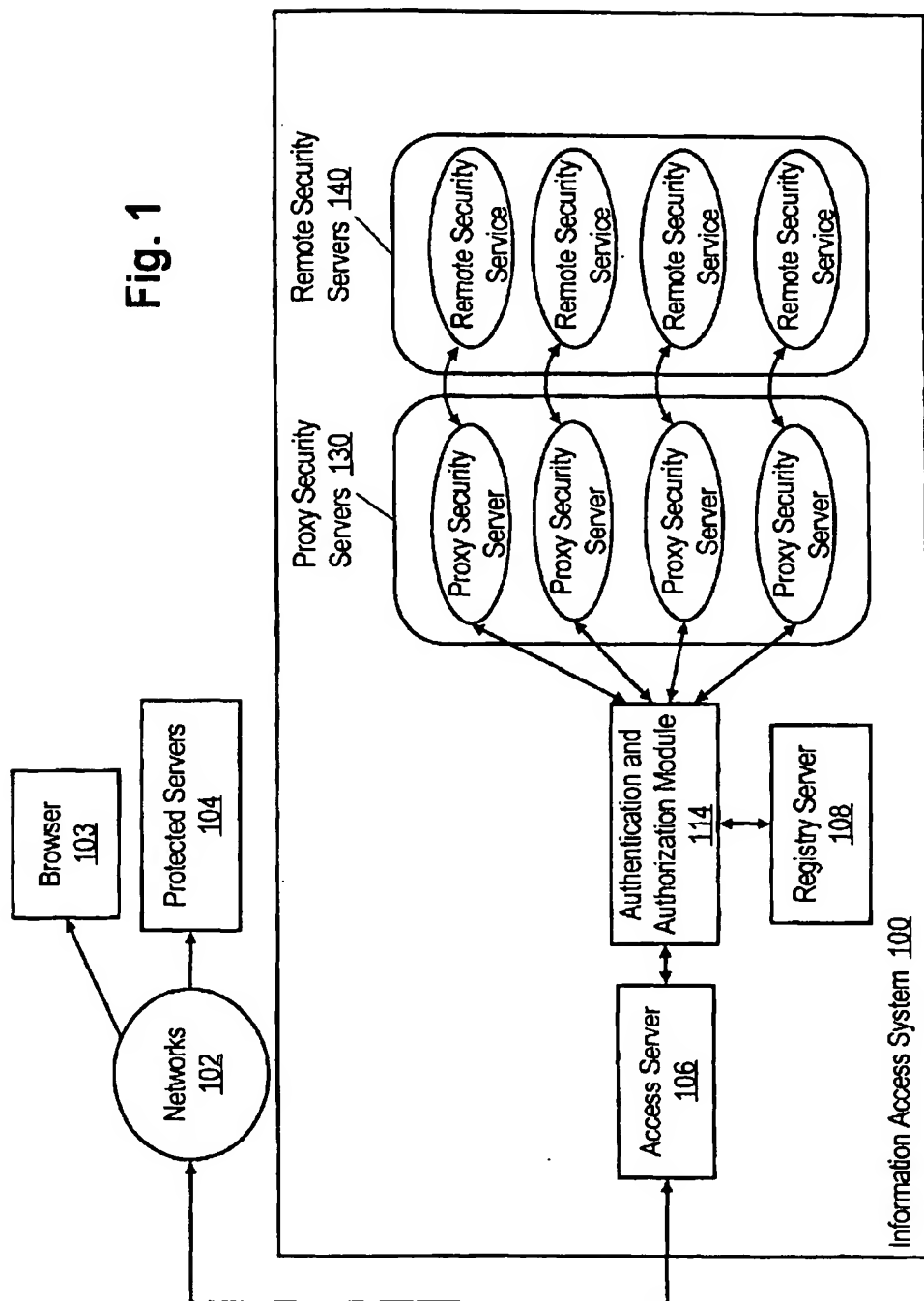


Fig. 1



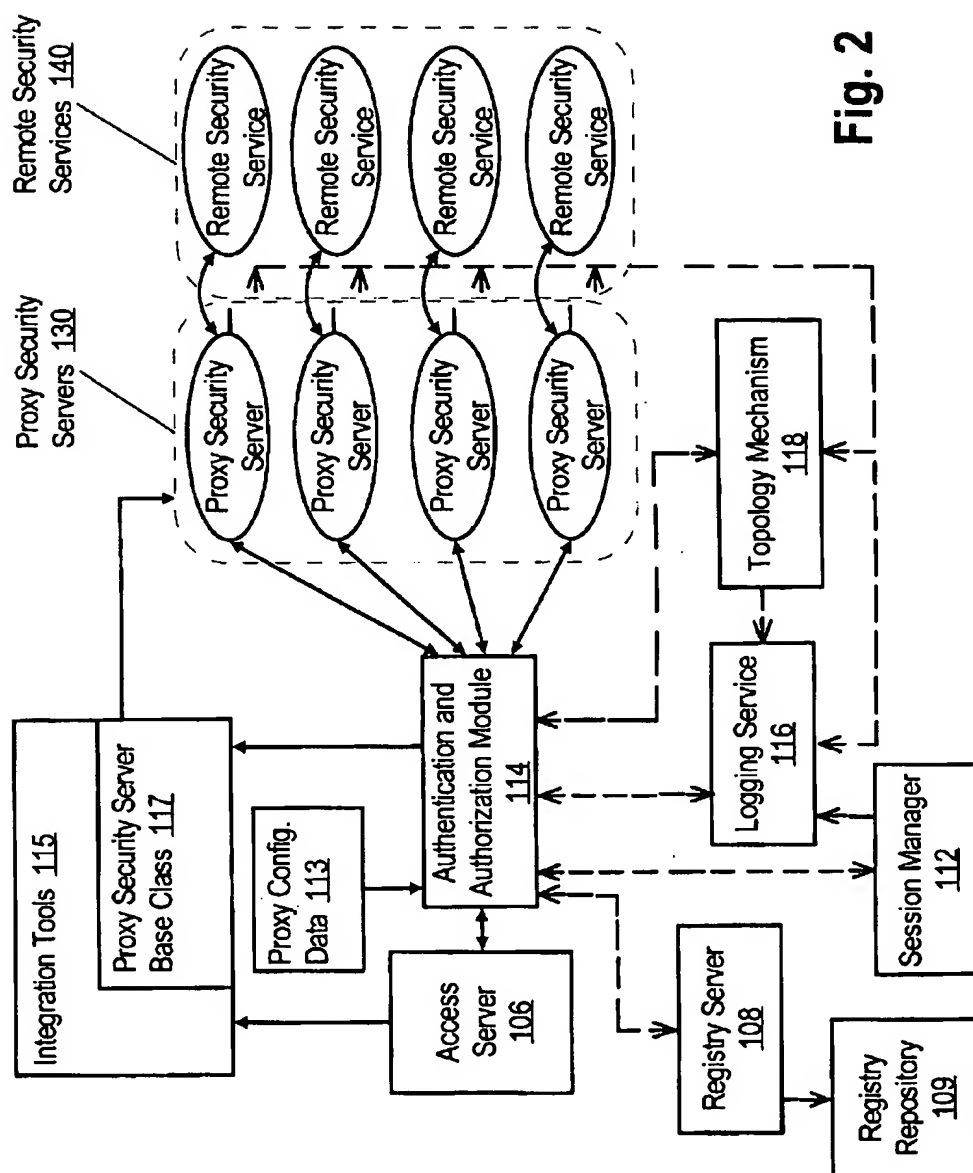
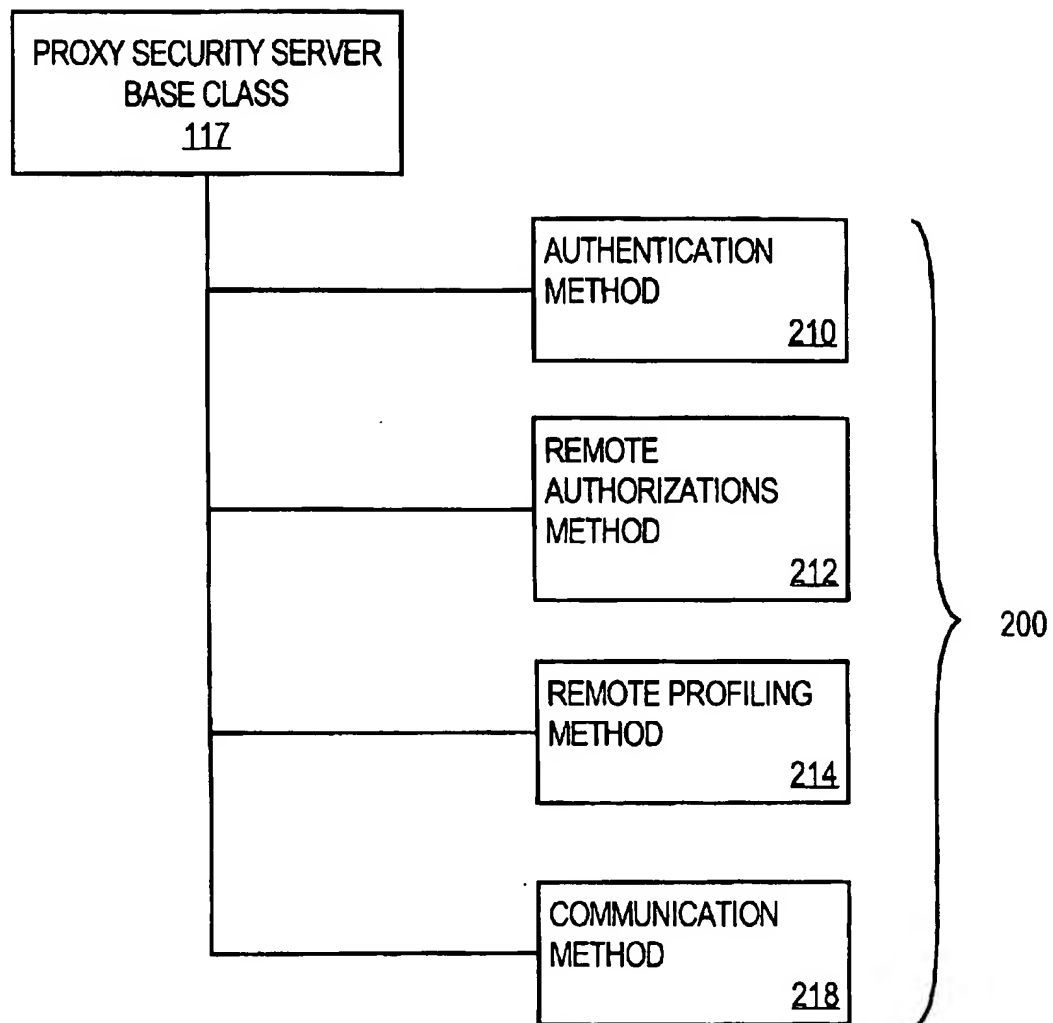
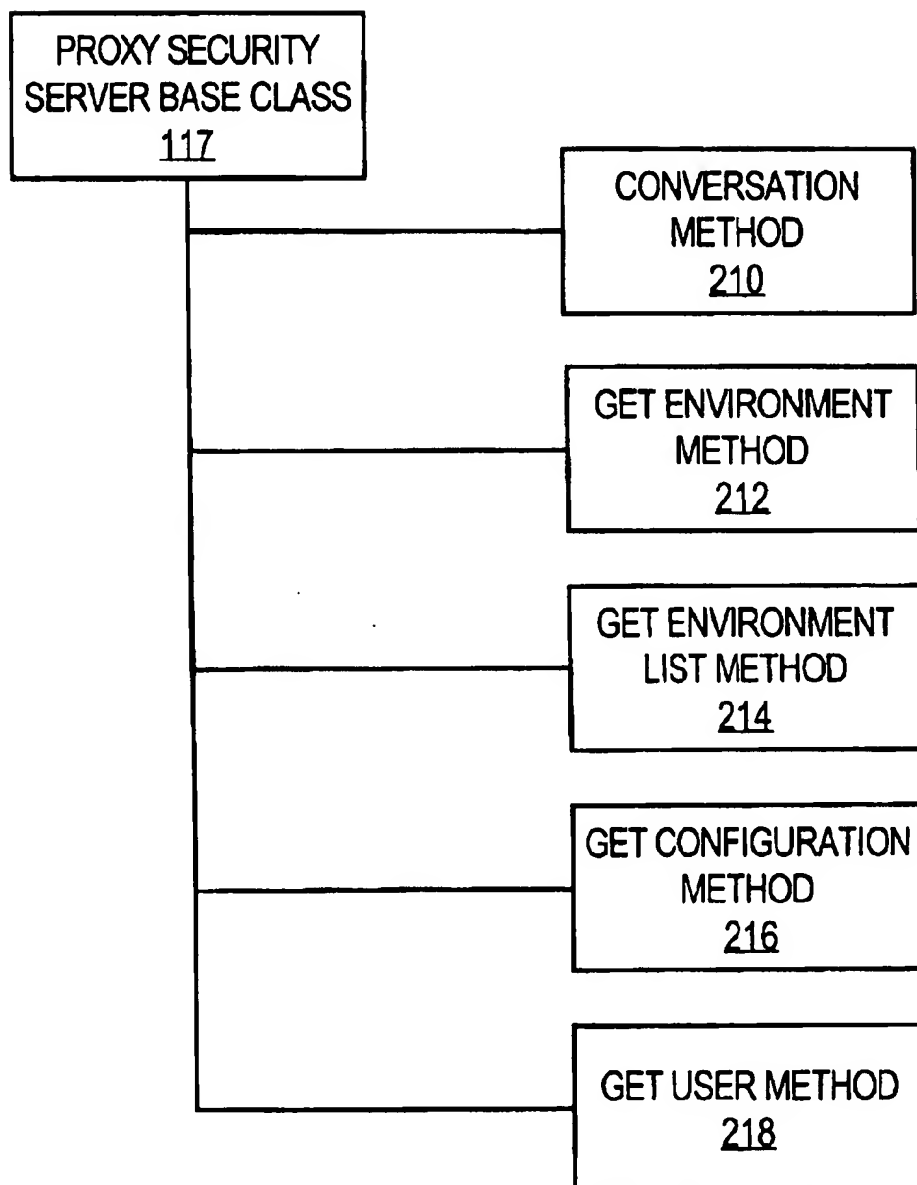


Fig. 2

**FIG. 3A**

**FIG. 3B**

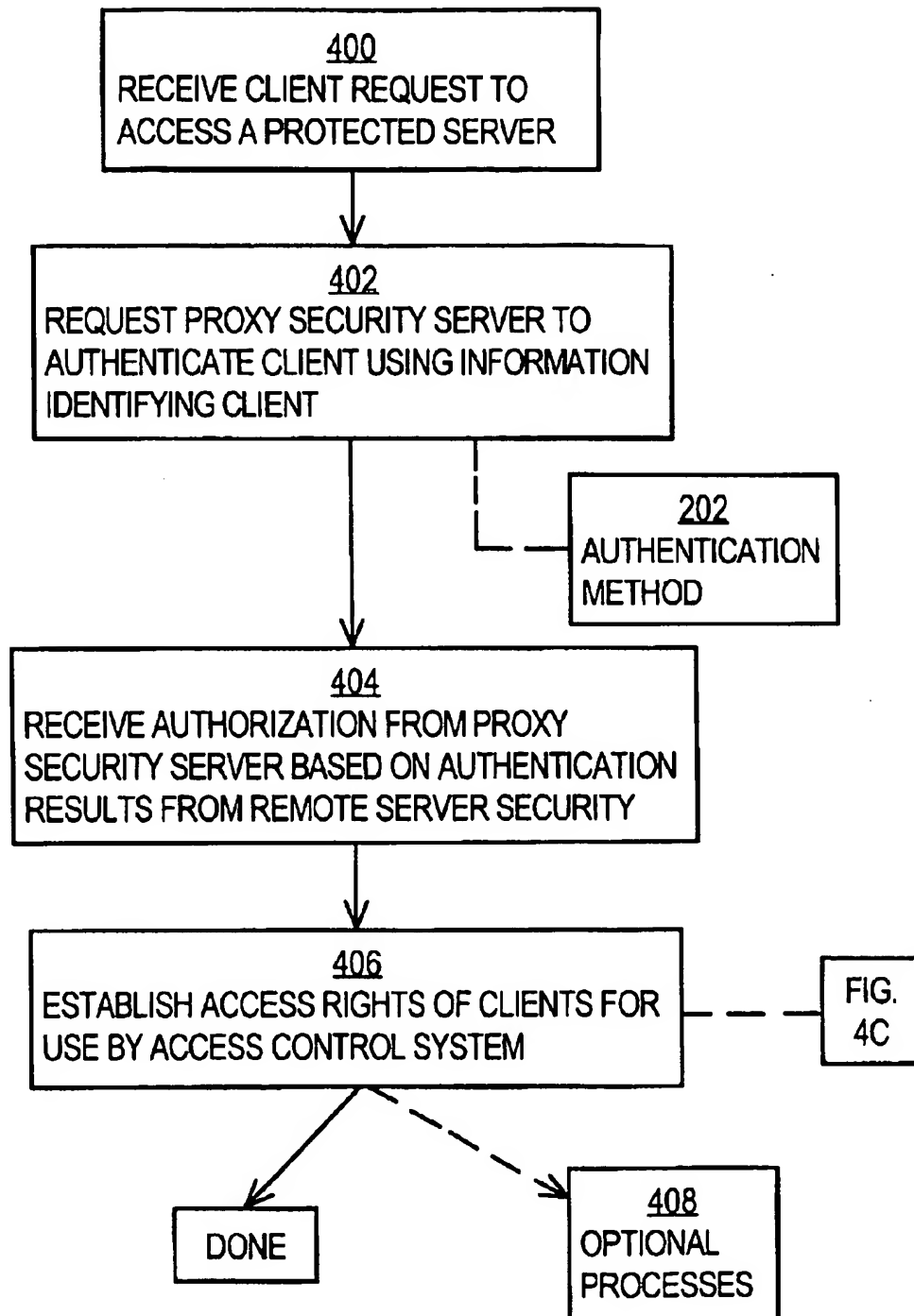
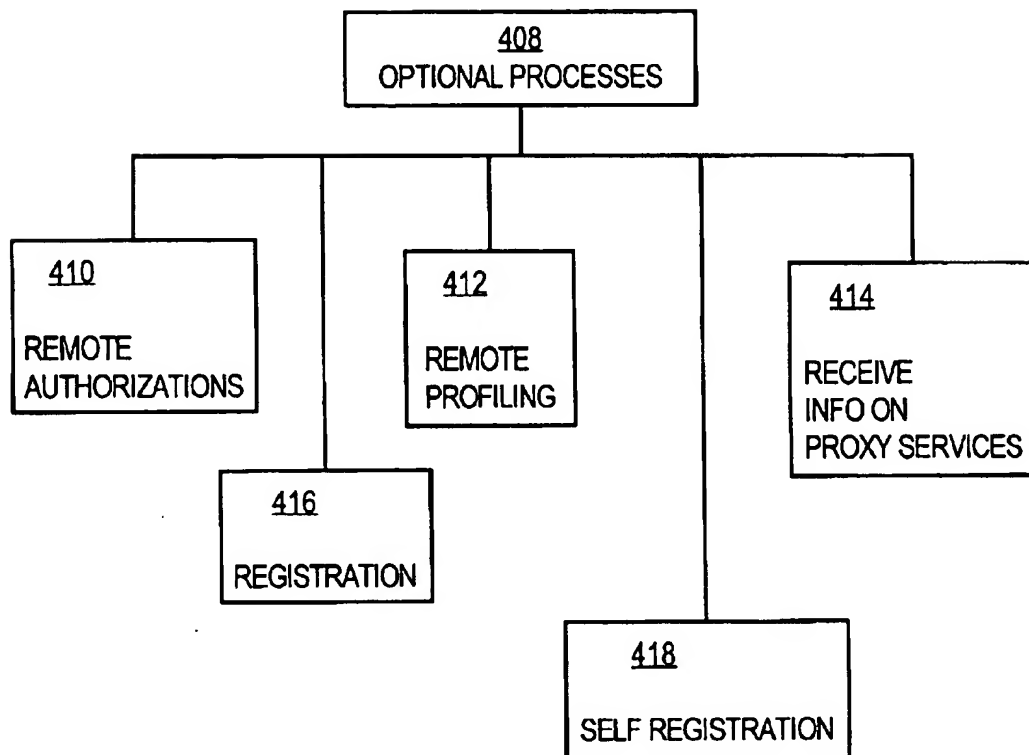
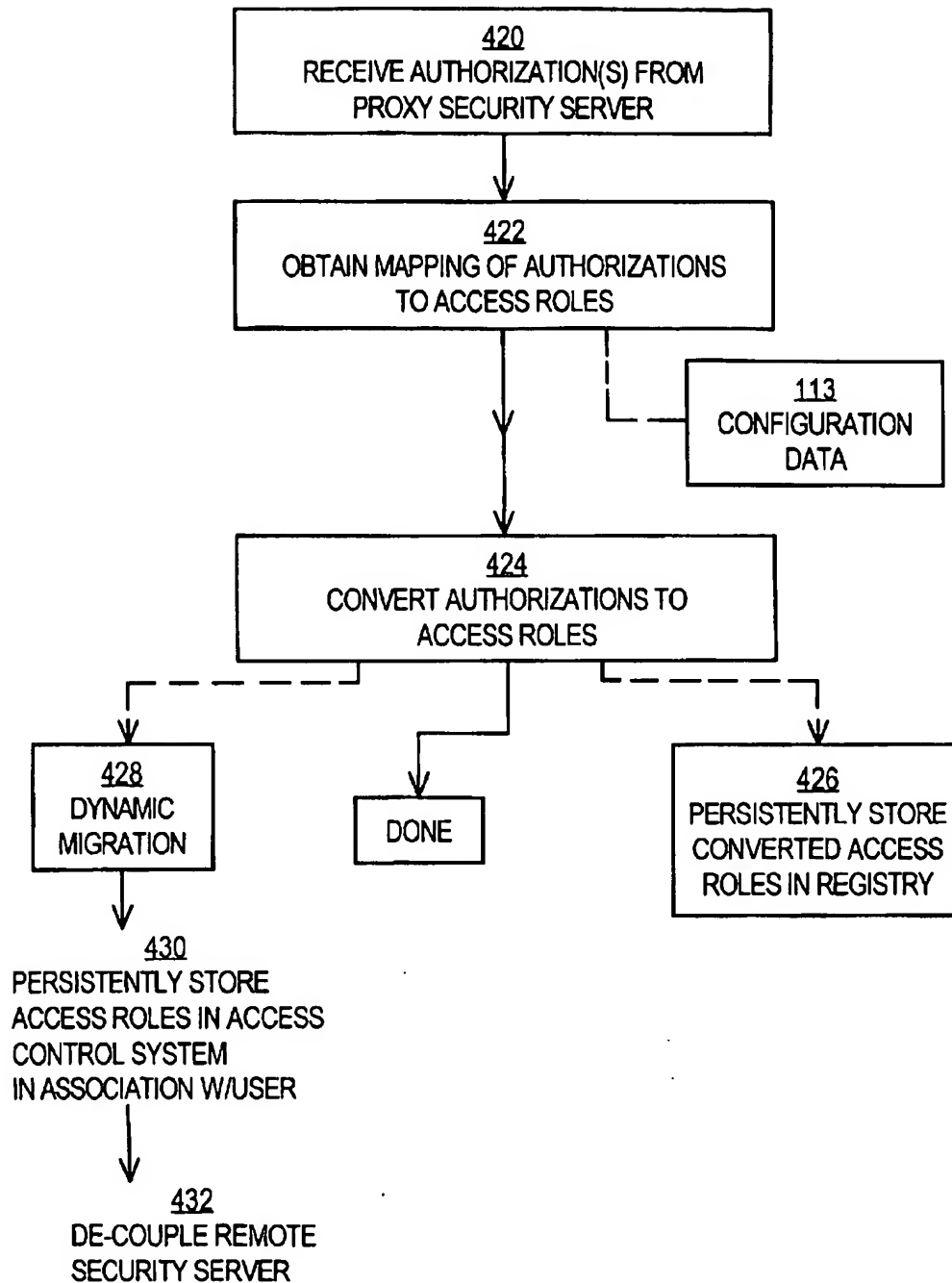
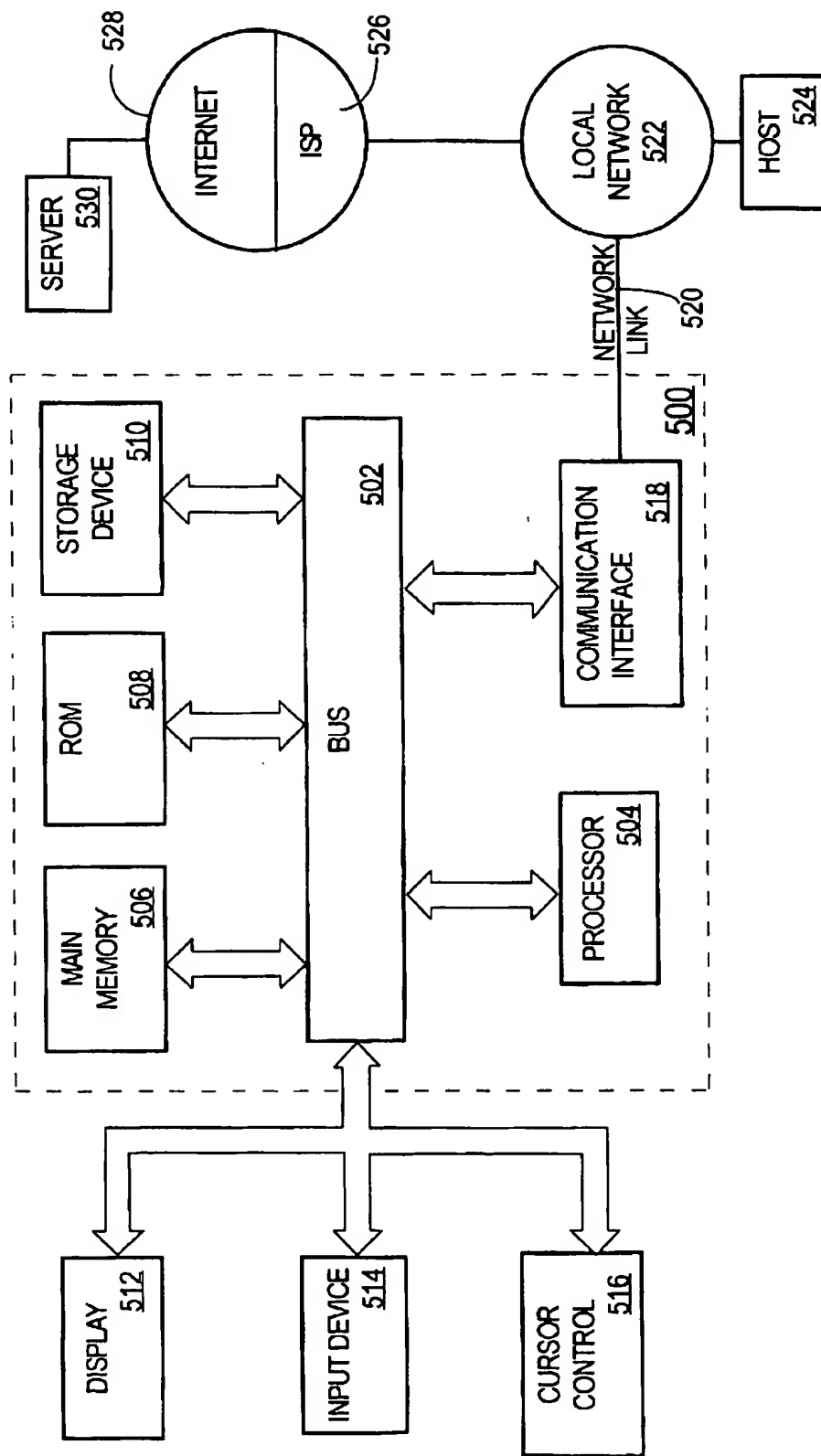
**FIG. 4A**



FIG. 4B



**FIG. 4C**

**FIG. 5**

1

# INTEGRATING HETEROGENEOUS AUTHENTICATION AND AUTHORIZATION MECHANISMS INTO AN APPLICATION ACCESS CONTROL SYSTEM

## FIELD OF THE INVENTION

The present invention relates to security systems in computer systems, and in particular, integration of heterogeneous security systems into an application access control system.

## BACKGROUND OF THE INVENTION

Computer networks have become ubiquitous in business, industry, and education. These networks typically have one or more resources, such as applications, that provide various computing functions. Development of the globally accessible, packet-switched network known as the Internet has enabled network resources to become available worldwide. Hypertext protocols that implement the World Wide Web have evolved, ("The Web"), enabling networks to serve as a platform for global electronic commerce and the easy exchange of information between businesses and their customers, suppliers and partners.

Businesses are rushing to make their applications available over networks, including the Web, and just as quickly stumbling into several roadblocks. For example, some information is valuable and sensitive, and needs to be made available only to selected users. Thus, there is a need to provide selective access to network resources and information over the Web.

This need exists in the context of internal Web networks that are available to employees of an organization, called Intranets, as well as Web networks and resources that are available to external customers, suppliers and partners of the organization, called extranets. Extranet users may require access to a large number of applications, for example, product catalogs, customer databases, or inventory systems. There may be millions of potential users, the number of which grows dramatically as an organization grows.

One approach to some of the foregoing problems and needs is the application approach. Under the application approach, a security mechanism is provided for each application program. Often, the security mechanism provided for an application is the application's own native security system. When a user connects to an application through a network, the security mechanism for the application is invoked. For example, when connecting to an accounting application, the accounting application invokes its security mechanism. The security mechanism obtains a user id and password from the user, and then authenticates the user. Authentication refers to the process of using information to identify a user ("authentication input") and verifying that the user is what the information purports the user to be. Examples of authentication input include user id and password received from a user, or a digital certificate.

An advantage of the application approach is that it may use security mechanisms that already exist. Use of existing security systems avoids reprogramming applications to use another security system and reconfiguring the other security system, by, for example, re-entering the user id and passwords of existing users.

A disadvantage of the application approach is that it results in a heterogeneous set of security mechanisms, each of which may present the user with a different authentication

2

procedure. Even if two security systems use the same authentication procedure, such as user id/password authentication, a user may use one user id and password pair on one system, and another user id and password pair on another system. Obviously, tracking different user ids and passwords can be very burdensome to a user.

Another disadvantage of the foregoing approach is duplication of management processes. To provide user access to a set of applications, an administrator must repeatedly add the user to each security system in use. The redundancy of these processes, combined with rapid growth in the number of users, can make the cost of deploying, managing and supporting a system unacceptably high.

Another disadvantage stems from the use of a common user interface for accessing applications over a network. The user interface is configured to interact with each security mechanism that may be accessed through the common user interface. Thus, adding a new security mechanism for a new or existing application may require reprogramming, recompilation, and reinstallation of the common user interface.

For example, new security mechanisms such as retinal scanners are becoming available. However, integrating such mechanisms is difficult. The required effort may increase costs and delays to implement new applications and security mechanisms to undesirably high levels.

Based on the foregoing, it is clearly desirable to provide a mechanism to govern access to one or more information resources in which selective access is given to particular users, a mechanism that is equally adaptable to an internal network environment and to an external network environment and which takes advantage of existing security mechanisms, and a mechanism that is easy to re-configure as new user applications and authentication techniques become available.

## SUMMARY OF THE INVENTION

The foregoing needs and objects, and other needs and objects that will become apparent from the following description, are achieved by the present invention, which comprises, in one aspect, an access control system. The access control system includes a server which provides authentication and authorization services. The server uses the authentication and authorization services from a set of remote servers, which may be servers that provide the authentication services from legacy access control systems, or specialized access control systems such as authentication services based on retinal scans.

The services of the remote servers may be accessed through proxy servers. The proxy server serves as an interface between the server and the other remote servers, and provides an API through which the services of the remote servers may be accessed. The proxy servers may be instantiations of a subclass of a base class. The base class defines methods for the API. Due to the power and simplicity of the inheritance feature of object oriented technology, developers may develop subclasses which inherit the methods of the base class. A software developer need only implement methods needed to interface with a particular remote server.

A remote server may provide authentication services, authorization services, and the ability to edit information stored on the remote servers regarding users. The authorizations received by a server from the remote server may be translated into a form of authorizations used by the server. The translated authorizations may be migrated to the server, and stored in persistent storage for later use by the server.

3

## BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

FIG. 1 is a block diagram depicting an access control system coupled to a system protected by the access control system;

FIG. 2 is a block diagram depicting an access control system in greater detail than FIG. 1;

FIG. 3A is a block diagram of method that may be used to implement an integration of security systems in an application access system;

FIG. 3B is a block diagram of additional methods;

FIG. 4A is a flow diagram of a method of integrating security systems into an application access system;

FIG. 4B is a flow diagram of optional processes in the method of FIG. 4A;

FIG. 4C is a flow diagram of further steps in the method of FIG. 4A;

FIG. 5 is a block diagram depicting a computer system upon which an embodiment of the present invention may be implemented.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A method and apparatus for integrating heterogeneous authentication and authorization mechanisms into an application access control system is described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

## Operational Context

FIG. 1 is a block diagram depicting elements of an information access system 100 according to a preferred embodiment. Generally, an Information Access System 100 comprises a plurality of components including an Access Server 106, Registry Server 108, and Proxy Security Servers 130, and Remote Security Servers 140. The foregoing components cooperate to control access to resources stored on one or more Protected Servers 104. Each component comprises one or more modules. There may be any number of Protected Servers 104. Users are individuals who have a relationship with an organization and play various roles, and are registered in the system 100. Users may be members of an organization, or may be customers, suppliers, or business partners of the organization. Administrators control the system. A server is a executable module that resides on at least one computer and which provides services to clients requesting those services.

In one embodiment, Protected Servers 104, Access Server 106, and Registry Server 108 are configured as disclosed in co-pending application Ser. No. 09/113,609 filed Jul. 10, 1998, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

A browser 103 is coupled by a communication link to a network 102. The block shown for browser 103 represents a terminal, workstation computer, or an equivalent that executes a standard Web browser program or an equivalent, such as Netscape Communicator or Internet Explorer. Net-

4

work 102 is a compatible information communication network, such as the Internet. In alternate embodiments, the browser 103 is a client process or client workstation of any convenient type, and the network 102 is a data communication network that can transfer information between the client and a server that is also coupled to the network.

Authentication and Authorization Module 114 manages authentication and authorization services in Information Access System 100. To provide authentication and authorization services, Authentication and Authorization Module 114 uses data stored in a central repository and managed by Authentication and Authorization Module 114 or uses authentication and authorization services provided by Remote Security Servers 140. The authentication and authorization services provided by Remote Security Server 140 may be accessed through a Proxy Security Server 130. A Proxy Security Server 130 serves as an interface between Authentication and Authorization Module 114 and a Remote Security Server 140.

Proxy Security Servers 130 provide an application programmer interface ("API"). An API is a symbolic interface that defines inputs and outputs to a set of computer program routines through which services provided by a server may be accessed by clients of the server. The API provided by Proxy Security Servers 130 allows Authentication and Authorization Module 114 to access services provided by Remote Security Servers 140. To assist Remote Security Servers 140 in servicing the requests of Information Access System 100, Authentication and Authorization Module 114 also provides a set of services for Remote Security Servers 140. These services are also accessed through the API.

Each of Remote Security Servers 140 manages authentication profiles about users and may manage information about access rights of one or more applications residing on Protected Servers 104. Access rights refer generally to actions that may be performed on behalf of a given user, including, for example, access to read or write data from a Resource. A Remote Security Server 140 may be, for example, a legacy security service used by an accounting application that resides on a Protected Server 104, which may be running under the Windows NT™ operating system. The legacy security service is the security service provided by Windows NT. Authentication and Authorization Module 114 may access the security service of Windows NT through a Proxy Security Server 130. This ability enables Information Access System 100 to manage access to the accounting application through information and services that already exist on other security systems, preserving time, effort, and money already invested in those systems.

## Access Roles

The Information Access System 100 enables administrators to implement access rules by defining roles (Access Roles) that users perform when working for an organization or conducting business with an enterprise. An Access Role may reflect a relationship of a user to the organization (employee, customer, distributor, supplier), their department within an organization (sales, marketing, engineering) or any other affiliation or function (member of quality task force, hotline staff member) that defines their information needs, and thus their access rights or privileges. Examples of Roles include Employee, Customer, Distributor, Supplier, Sales, Marketing, Engineering, and Hotline Staff.

Access Roles are defined by information identifying a name of a role and by a functional group in which the role resides. A functional group is often a department in which similar functions exist. Examples of functional groups are Marketing, Sales, Engineering, Human Resources, and Operations. Access Roles are also associated with a User Type.

Access Roles determine what resources a User can access. Further, each Access Role may allow access to a subset of information that is available in resources. For example, a User who is an Employee in the Marketing department could access Price List and New Products Resources. Thus, Information Access System 100 enables the creation of resource profiles by assigning roles to resources, and by assigning roles to users to generate access rights. Information Access System 100 automatically links a user to the resources profiles that have been assigned the same roles, so that deployment of new resources may occur rapidly.

Information Access System 100 may manage such roles using the methods and mechanisms disclosed in co-pending application Ser. No. 09/248,764, filed Feb. 12, 1999, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

#### User Login

The Information Access System 100 also enables Users to log-in to the system once, and thereafter access one or more resources on a network during an authenticated session. Users may, for example, log in either with a digital certificate or by opening a login page URL with a web browser and entering a name and password. In the past, users have had to log in individually to each application that they are authorized to use. In the preferred embodiment, users always access the same login page regardless of the number of resources to which they need access. Thus, the system provides a mechanism of single secure log-in to resources available on a network.

When a user attempts log in, Information Access System 100 establishes a session. Specifically, a unique session number is generated, and port and connection information ("session information") is stored in association with the session number. The combination of a session number and session information is referred to as a session. Session information includes information that identifies a user, and that indicates whether the user has been authenticated. A session associated with a user that has been authenticated is referred to as an authenticated session until the authenticated session expires. An authenticated session may, for example, expire after the elapse of threshold period of time. Until the authenticated session expires, the user associated with the authenticated session may access one or more resources protected by the Information Access System 100.

If the Login attempt is successful, that is, the user has been authenticated and is authorized to access resources protected by Information Access System 100, the Information Access System 100 presents the user with a personalized menu. The personalized menu assists the User in identifying and selecting a resource to access. In one embodiment, a personalized menu is an HTML page containing a list of authorized resources. The personalized menu displays only resources to which the User has access. The user can then select and access a Resource.

Session information may be structured and managed using the methods and mechanisms disclosed in co-pending application Ser. No. 09/363,315, filed Jul. 28, 1999, the entire disclosure of which is hereby incorporated by reference as if fully set forth herein.

#### Integrating Authentication Mechanisms

FIG. 2 shows Information Access System 100 in greater detail. Access Server 106 stores a log-in page, and is coupled to the Authentication and Authorization Module 114. The Access Server 106 may receive authentication input regarding a particular user for a particular session and transmit it to the Authentication and Authorization Module 114, which uses the information to authenticate a user. The Authentica-

tion and Authorization Module 114 returns information that specifies whether the user is authenticated, and the access roles for the user.

The Authentication and Authorization Module 114 authenticates the user in one or more ways. The Authentication and Authorization Module 114 may authenticate a user by verifying the password with the Registry Server 108. Alternatively, the Authentication and Authorization Module 114 may use authentication services of one of the Remote Security Services 140 to authenticate the user.

The Registry Server 108 manages a registry repository 109, which may be structured as a database. The registry repository contains information about how to authenticate users and what authorizations a user has, including a mapping of a particular user to access roles. Information about how to authenticate a particular user is referred to as an authentication profile, and may include data indicating a user id, password, and which Proxy Security Servers 130 to use to authenticate a user. An authorization is a privilege to perform a particular action with respect to a set of resources on a computer system. For example, an authorization may define access to a particular Web page or a set of directories.

For each user it authenticates, the Authentication and Authorization Module 114 generates data representing the authorizations of the user in the form of access roles. The Authentication and Authorization Module 114 generates the access roles using authorization information recorded by Registry Server 108, or authorization information obtained from Remote Security Service 140 through a Proxy Security Server 130. It then encrypts data representing the access roles and sends the encrypted data in a cookie to the user's browser. A "cookie" is a packet of data sent to web browsers. Each cookie is saved by browser 103 until the cookie expires. A returned cookie is required for access to resources protected by Information Access System 100.

A Session Manager 112 manages sessions throughout Information Access System 100. Session Manager 112 establishes new sessions upon request by various components of Information Access System 100, including, for example, Access Server 106 when Access Server 106 is logging in a user. Session Manager 112 also expires sessions according to various criteria and techniques.

Logging Service 116 receives information about the actions taken by various modules of Information Access System 100 and records actions as events in one or more logs. For example, when Session Manager 112 expires an authenticated session, Session Manager 112 transmits a message to record the expiration of the authenticated session.

The Topology Mechanism 118 tracks various components of Information Access System 100 when the components starts up. In particular, Topology Mechanism 118 receives messages from a particular component when it starts up, and, in response, may transmit messages to other components to inform them that the particular component has started.

Integration Tools 115 are selectively executed on Information Access System 100 and function to customize the particular configuration of the foregoing components. For example, Integration Tools 115 are used to customize the form or content of screen displays presented by browser 103 for user login and logout, or to enforce password selection rules, or to program and develop Proxy Security Servers 130. Integration Tools 115 include Proxy Security Server Base Class 117, which are later described.

Proxy Configuration Data 113 is data that specifies the configuration of each of Proxy Security Servers 130. Proxy

Configuration Data 113 specifies, for example, whether a particular Proxy Security Server 130 provides authorization services. Proxy Configuration Data 113 shall later be described in greater detail.

In one embodiment, Information Access System 100 is stored on and executed by one physical server or computer. In alternate embodiments, one or more components are installed on separate computers. Starting Up the Authentication and Authorization Module and Proxy Security Servers

In an embodiment, the Session Manager 112, Logging Service 116, and Topology Mechanism 118 start up before Authentication and Authorization Module 114 and any of Proxy Security Servers 130. Relative to each other, a Proxy Security Server 130 and the Authentication and Authorization Module 114 may start-up in any order.

When the Authentication and Authorization Module 114 starts up, it reads Proxy Configuration Data 113 and stores data representing the Proxy Configuration Data 113 in memory. Authentication and Authorization Module 114 then checks-in with the Topology Mechanism 118. If any of Proxy Security Servers 130 have previously started up and checked-in with Topology Mechanism 118, then Topology Mechanism 118 transmits a message to Authentication and Authorization Module 114 indicating the availability of the already running Proxy Security Servers 130.

When a Proxy Security Server 130 starts up, it checks-in with Topology Mechanism 118. If Authentication and Authorization Module 114 is already running, then Topology Mechanism 118 transmits a message indicating the availability of the Proxy Security Server 130 that has just checked in.

#### Proxy Security Service Framework

A Proxy Security Server 130 allows Authentication and Authorization Module 114 to access any Remote Security Servers 140 through a common API. The API enables Authentication and Authorization Module 114 to interface with any Remote Security Servers 140 in a common manner. The Authentication and Authorization Module 114 and Proxy Security Servers 130 may be compliant with Common Object Request Broker Architecture (CORBA) communicating to each other using the framework defined by CORBA. Proxy Security Servers 130 may be Java objects that are instantiations of Java classes.

Proxy Security Server Base Class 117 is an abstract class from which Proxy Security Server Subclasses inherit methods and data. A Proxy Security Server Subclass, represented in FIG. 2 by Proxy Security Servers 130, is a class from which a Proxy Security Server 130 may be instantiated. The Proxy Security Server Base Class 117 defines methods of the API. Some of the methods are implemented in the base class. Other methods are implemented in a subclass of the base class.

A Proxy Security Server 130 may implemented as multi-threaded CORBA server that interacts with the Authentication and Authorization Module 114 and a Remote Security Servers 140. The Proxy Security Servers Base Class 117 supplies the implementations to support CORBA, multi-threading, and other aspects of a Proxy Security Server 130. Due to the power and simplicity of inheritance in object oriented development environment, a developer of a particular Proxy Security Server subclass does not have to address CORBA or the multi-threaded aspects of a Proxy Security Server subclass. A developer need only extend from the Proxy Security Server Base Class 117 to implement a limited set of methods needed for a particular Remote Security Server 140.

#### Services Provided by Proxy Security Servers Through the Common API

A Proxy Security Server 130 makes available services provided by Remote Security Servers 140 to Information Access System 100. These services are accessed by invoking a method of the API, and in particular, may be accessed by invoking methods of Proxy Security Servers 130. The methods may be implemented in the Proxy Security Server Subclasses 200, as shown in FIG. 3A, although for some of these methods a default behavior is implemented in the base class. A description of the methods that may be implemented, and the functionality they provide as services, follows. Additional details about these methods and their functionality, and about additional methods and functionality, may be found in Appendix I.

**AUTHENTICATION:** A Proxy Security Server Base Class 117 defines at least one method that may be invoked to authenticate a user. The authentication method 202 is implemented in a Proxy Security Server Subclass. See, for example, `pam_sm_authenticate` in Appendix I. Generally, the implementation for this method is specific to a particular Remote Security Server 140. Authentication and Authorization Module 114 passes a session identifier and authentication input, such as a user id and password, digital signature, or biometric data, for example, a thumbprint image. The method is executed to authenticate the user associated with the session, in accordance with its then-current implementation. The architecture and configuration of elements disclosed herein, in combination with the power and versatility of object oriented languages that may be exploited to develop subclasses results in a set of Proxy Security Servers 130 that interface to a wide range of Remote Security Servers 140.

**REMOTE AUTHORIZATION:** Proxy Security Server Base Class 117 may define at least one method that allows Authentication and Authorization Module 114 to modify authorizations on a Remote Security Server 140. The remote authorization method 204 is implemented in a Proxy Security Server Subclass. See, for example, `pam_chauthtok` (String `pamh`), in Appendix I. The ability to modify authorizations on a Remote Security Server 140 is referred to as remote authorizations. A developer provides the implementation for the method. When Authentication and Authorization Module 114 invokes the method, it passes in a session identifier.

**REMOTE PROFILING:** Proxy Security Server Base Class 117 defines at least one method that provides the ability to modify a user profile on a Remote Security Server 140. A user profile is data describing demographic information about a user, for example, the user's address, full name, and marital status. The method remote profiling 206 is implemented in a Proxy Security Server Subclass. See, for example, `pam_chauthtok`, in Appendix I. This ability to alter a user profile on a Remote Security Server 140 is referred to as remote profiling. A developer provides the implementation for the method. When Authentication and Authorization Module 114 invokes the method, it passes in the session identifier.

**COMMUNICATING AVAILABLE FUNCTIONALITY:** Not all Proxy Security Servers 130 provide the same set of services. For example, a Proxy Security Server 130 may provide authentication and remote profiling, but not remote authorization. When a Proxy Security Server 130 starts up, Authentication and Authorization Module 114 needs to know what services a particular Proxy Security Server 130 provides. For this purpose, a Proxy Security Server Base Class 117 provides at least one communication method. The

communication method 208 is implemented in a Proxy Security Server Subclass. See, for example, `getImplementedFunctions()` in Appendix I.

Functionality Provided by Authentication and Authorization Module to Proxy Security Servers

To provide services to Authentication and Authorization Module 114, a Proxy Security Server 130 may need to obtain information through Authentication and Authorization Module 114, or request other services from Authentication and Authorization Module 114. For example, a Remote Security Server 140 may need to query a user to get data for changing a user profile. Such services are accessed by invoking a method of the API defined in Proxy Security Server Base Class 117. A description of these services follows and FIG. 3B depicts the structure of the methods. Additional details about these services and their functionality, and about additional services and methods, may be found in Appendix II.

**INTERFACING WITH A HUMAN USER:** Proxy Security Server Base Class 117 defines at least one method that be invoked to communicate with a human user. Such methods are referred herein to as conversation methods. See `pam_conv` in Appendix II. For example, to change a user profile, a method that implements remote profiling may invoke a conversation method 210 to obtain inputs from the user that are used to update the user profile. Input parameters passed to the conversation method may specify whether user input should be obtained using a text box, a label for the text box, and whether user input is echoed as it is entered, or a list of selections to be displayed in a list box presented to the human user.

**ENVIRONMENT INFORMATION:** Proxy Security Server Base Class 117 defines at least one method that may be invoked to obtain information about a user or the operating environment. For example, a method may be invoked to obtain the user id and password associated with a session, or the network address of the device from which the user is attempting to log in. These methods may comprise a Get Environment method 212, Get Environment List method 214, Get Configuration method 216, and Get User Method 218. See, `pam_getenv`, `pam_getenvlist`, `pam_get_configuration`, `pam_get_user`, for example, in Appendix II.

**Registration and Self Registration**

Registration refers to a process of receiving data that identifies a user and specifies an authentication profile for the user. A user may be registered in Information Access System 100 in a variety of ways. For example, a system administrator may use an administrative user interface provided by Information Access System 100 for receiving data used to register users from a system administrator. In particular, Information Access System 100 receives input that specifies a user id, which Information Access System 100 stores in Register Server 108. In addition, Information Access System 100 receives data from a system administrator specifying an authentication profile. The authentication profile may include data specifying a Proxy Security Server Name that corresponds to a Proxy Security Server 130 used to authenticate the user.

Self-Registration refers to a process in which a user registers itself, by supplying to the Information Access System 100 data that specifies an authentication profile. For example, when a user first logs into Information Access System 100, Access Server 106 presents a login page to the user. The login page includes a text box for the user to enter a user id and a password, and a list box listing Proxy Security Server Names as selections. The user enters the strings "John Doe" as the user id and "DoeString" as the password, and picks a selection from the list box. Informa-

tion Access System 100 then transmits a message to a Proxy Security Server 130 that corresponds to the selected Proxy Security Server Name. In an embodiment, the message is transmitted by invoking the `pam_auth` method of the Proxy Security Server 130.

The Proxy Server 130 then transmits a message indicating that the user is authenticated to Authentication and Authorization Module 114. In response, Authentication and Authorization Module 114 registers the user id, storing in Registry Server 108 data specifying the user id and Proxy Security Server Name. The user may later add or delete Proxy Security Server Names, or other information in the user's authentication profile.

**Dynamic Authorization Mapping**

Once a registered user is authenticated, the authorizations for the user are determined. Typically, a user is associated with a default set of authorizations. In addition, a system administrator may have assigned access roles to a particular user. When a user logs into Information Access System 100, the Authentication and Authorization Module 114 gets from Registry Server 108 the access roles assigned to the user. In addition, the Authentication and Authorization Module 114 will get authorizations from any Proxy Security Server 130 specified in the user's authentication profile.

The authorizations obtained from Proxy Security Server 130 may not be in the form of access roles, and thus may be converted to Access Roles in a process referred to as Dynamic Authorization Mapping. In a dynamic authorization mapping, authorizations provided by Proxy Security Servers 130 are converted to Access Roles based on a mapping. The mapping is stored in Proxy Configuration Data 113, and shall be described in greater detail. The Access Roles generated by the conversion are merged with any Access Roles specified by Registry Server 108 for the user.

**Migrating Authorizations from Remote Security Servers**

Information Access System 100 may be used to replace a legacy security system. The effort invested in configuring a legacy security system with authentication and authorization information may be preserved by implementing a Proxy Security Servers 130 for the legacy security system. However, it may be desired to eventually discontinue use of the legacy system, so the information on the legacy security system must be transferred to Information Access System 100.

One technique described herein for efficiently transferring this information is referred to Dynamic Migration. Dynamic Migration is the process of permanently assigning Access Roles generated by converting information from a Remote Security Server 140. These converted access roles are subsequently associated with the user whenever the user logs in. Dynamic Migration may be performed when a user self registers. Data in Proxy Configuration Data 113 specifies whether Dynamic Migration is performed for a Proxy Security Server 130, that is, whether Dynamic Migration is performed for a user who self registers and selects the Proxy Security Server 130.

**Configuration Data**

Configuration Data 113 is organized as one or more blocks of data, each of which is associated with a Proxy Security Servers 130. Each block contains entries that specify an operational aspect of a Proxy Security Server 130. Each entry contains a string specifying a parameter name, followed by a '=' character, followed by string specifying a parameter value. An example of Configuration File 15 follows.



11

```

[UserPasswordPam]
name=UserPasswordPam
desc=Userid/Password
usrtag=Userid
pwdtag=Password
env=
translate=false
selfreg=false
authsourcename=UserPassword
config =
  getAccessRoot:c:\enCommerce\getAccess|debug:false
[CertPam]
name=CertPam
desc=Certificate
usrtag=
pwdtag=
env=CLIENT_CERT|COOKIE:COOKIE#
translate=false
selfreg=false
authsourcename=Certificate
config =
  getAccessRoot:c:\enCommerce\getAccess|debug:false

```

A block begins with a tag, which may be a bracketed string, that specifies a Proxy Security Server Name. The example above has two blocks. The first block begins with the tag '[UserPasswordPam]'.

The first entry in the first block is the name value pair 'name=UserPassword', and it specifies a Proxy Security Server Name. 'name' is the parameter name, and 'UserPassword' is the Proxy Security Server Name.

The Proxy Security Server Name is used to associate a Proxy Security Server 130 with a block. A Proxy Security Server 130 is associated with a Proxy Security Server Name when the Proxy Security Servers 130 is started, by, for example, specifying the Proxy Security Server Name as input argument value in the command used to invoke the proxy server. A Proxy Security Server 130 associated with a Proxy Security Server Name is herein referred to as the named Proxy Security Server 130.

The parameter 'desc' specifies a description that is displayed to the user to refer to the named Proxy Security Servers 130.

Parameter 'usrtag' is the entry for the label displayed next to a text box used to receive user input specifying a user id.

Parameter 'pwdtag' is the entry for the label displayed next to a text box used to receive user input specifying a password.

Parameter 'env' is used to specify an environmental variable that is caught. The block beginning with the tag [CertPam] indicates certification parameters and shows a use of the parameter 'env'. In this case, the environment variable could be called CLIENT\_CERT, or can be sent in the form of cookies.

The parameter 'selfreg' is used to specify whether a user may self register by selecting the named Proxy Security Server 130.

The parameter 'translate' specifies whether dynamic mapping is performed for the authorizations supplied by the named Proxy Security Server 130.

The 'config' entry depends on the Proxy Security Servers 130 itself. For example, a Proxy Security Servers 130 used to interface to a Remote Security Servers 140 that uses Windows NT Domain Authentication requires the number of domains to authenticate as well as the name of each of the domain.

12

Specifying a Mapping in the Configuration Data

To perform dynamic authorization mapping, a mapping is specified in the block associated with a Proxy Security Server 130. The example entry illustrates a mapping for a Proxy Security Server 130 that interfaces with a Remote Security Server 140 that uses Windows NT Domain Authentication.

```

[NTDomainPam]
name=NTDomainPam
desc=NT Domain Authentication
usrtag=Userid
pwdtag=
env=
translate=true
selfreg=true
authsourcename=NTDomain
config=numberOfDomains=1|domain1:Marketing
usertype=undef
Domain Users=emplye
Domain Admins=excacc.spradm

```

The example Proxy Configuration Data 113 above specifies to Authentication and Authorization Module 114 that a Proxy Security Server 130 associated with the Proxy Security Server Name NTDOMAIN can perform self-registration and dynamic authorization mapping. If the authorizations returned by the named Proxy Security Server 130 include the 'Domain Users' group, the user type will be translated to 'emplye'. Likewise, if the returned authorizations include 'Domain Admins', then the user has the administrative role referred to as 'excacc.spradm'.

Example Method of Operation

FIG. 4A is a flow diagram of an exemplary method of operation of the system of FIG. 2. The method of FIG. 4A may be implemented in one or more computer programs, processes, data structures, or related elements that form Authentication and Authorization Module 114.

In block 400, a request to access a protected server is received from a client. For example, Browser 103 as shown in FIG. 1 contacts Access Server 106 and requests an electronic document of one of the Protected Servers 104. Access Server 106 forwards the request to Authentication and Authorization Module 114.

In block 402, a proxy security server is requested to authenticate the client using information that identifies the client. For example, Authentication and Authorization Module 114 uses authentication method 202 to request one of the proxy security servers 130 to authenticate the client. In the course of executing the authentication step, the proxy security server may use one of the remote security services 140 to conduct authentication.

If the authentication request is unsuccessful, then the access server returns a message to the client indicating that access is refused. These steps are omitted from FIG. 4A for clarity. If the authentication is successful, then in block 404, authorization is received from the proxy security server based on authentication results that are received from a remote security service that is associated with the proxy security server.

In block 406, access rights for use with an application access system and associated with the client are established. The process of FIG. 4C may be used.

Optional processes may be carried out after block 406 or at any other appropriate time in the process, as indicated by block 408. FIG. 4B is a block diagram of optional processes 408 that may be carried out. The optional processes 408 may

include one or more of remote authorizations 410, remote profiling 412, or communications such as receiving information on available proxy services, as indicated by block 414. Optional processes 408 may also include registration 416 or self registration 418. These processes may be carried out using the mechanisms described above with respect to FIG. 2, FIG. 3A, and FIG. 3B.

FIG. 4C is a flow diagram of further steps that may be used to carry out the process of block 406.

In block 420, authorizations associated with the current client are received from the proxy security server. In block 422, a mapping of authorizations to access roles is obtained. The mapping may be stored in and obtained from Configuration Data 113.

In block 424, the authorizations are converted to access roles that can be used by an access control system such as Information Access System 100. When conversion is complete, one of three paths may be followed. One path is that the process may terminate processing, as indicated by the "DONE" block. Another path is that the process may persistently store the converted access roles in a registry, for example, using Registry Repository 108, for use later.

Still another path is that the process may carry out dynamic migration, as indicated by block 428. In dynamic migration, the access roles are persistently stored in the access control system in association with user identifying information. The remote security server may be decoupled from the system, as shown by block 432. Thereafter, the access roles are used to authenticate the user. In this way, authentication information in a legacy remote security server is automatically transferred to the access control system, and the legacy remote security server may be retired from service.

#### Hardware Overview

FIG. 5 is a block diagram that illustrates a computer system 500 upon which an embodiment of the invention may be implemented. Computer system 500 includes a bus 502 or other communication mechanism for communicating information, and a processor 504 coupled with bus 502 for processing information. Computer system 500 also includes a main memory 506, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 502 for storing information and instructions to be executed by processor 504. Main memory 506 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 504. Computer system 500 further includes a read only memory (ROM) 508 or other static storage device coupled to bus 502 for storing static information and instructions for processor 504. A storage device 510, such as a magnetic disk or optical disk, is provided and coupled to bus 502 for storing information and instructions.

Computer system 500 may be coupled via bus 502 to a display 512, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 514, including alphanumeric and other keys, is coupled to bus 502 for communicating information and command selections to processor 504. Another type of user input device is cursor control 516, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 504 and for controlling cursor movement on display 512. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

The invention is related to the use of computer system 500 as an access control system. According to one embodiment

of the invention, an access control system is provided by computer system 500 in response to processor 504 executing one or more sequences of one or more instructions contained in main memory 506. Such instructions may be read into main memory 506 from another computer-readable medium, such as storage device 510. Execution of the sequences of instructions contained in main memory 506 causes processor 504 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 504 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 510. Volatile media includes dynamic memory, such as main memory 506. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 502. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 504 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 500 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 502. Bus 502 carries the data to main memory 506, from which processor 504 retrieves and executes the instructions. The instructions received by main memory 506 may optionally be stored on storage device 510 either before or after execution by processor 504.

Computer system 500 also includes a communication interface 518 coupled to bus 502. Communication interface 518 provides a two-way data communication coupling to a network link 520 that is connected to a local network 522. For example, communication interface 518 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 518 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 518 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 520 typically provides data communication through one or more networks to other data devices. For

15

example, network link 520 may provide a connection through local network 522 to a host computer 524 or to data equipment operated by an Internet Service Provider (ISP) 526. ISP 526 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 528. Local network 522 and Internet 528 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 520 and through communication interface 518, which carry the digital data to and from computer system 500, are exemplary forms of carrier waves transporting the information.

Computer system 500 can send messages and receive data, including program code, through the network(s), network link 520 and communication interface 518. In the Internet example, a server 530 might transmit a requested code for an application program through Internet 528, ISP 526, local network 522 and communication interface 518. In accordance with the invention, one such downloaded application provides for an access control system as described herein.

The received code may be executed by processor 504 as it is received, and/or stored in storage device 510, or other non-volatile storage for later execution. In this manner, computer system 500 may obtain application code in the form of a carrier wave.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

#### APPENDIX I

Methods Used to Access Services Provided by Proxy Security Server

```
abstract protected int pam_sm_authenticate(String pamh, String args[ ])
```

Any Proxy Security Server Class should implement this method to provide authentication. This is the core of developing a Proxy Security Server Classes because authentication is the minimum action that a Proxy Security Server Classes performs. This method receives the Proxy Security Server Class handle (pamh) for the user session and a set of arguments. The Proxy Security Server Classes handle is the handle that provide extra information about the user, and the second parameter is an array of arguments that are passed to the authentication method. Currently, the second parameter is an empty array. This method should return one of the following return codes:

**PAM\_SUCCESS**—This code is returned when the authentication process was successful.

**PAM\_ACCT\_EXPIRED**—This code is returned when the user account is inactive, but the user is authentic.

**PAM\_USER\_UNKNOWN**—This code is returned when the user is an unknown user or the userid, password combination is not valid.

**PAM\_NEW\_AUTHTOK\_REQD**—This code is returned when a new password is required, this means that the current password has expired or is no longer valid for any other reason.

**PAM\_CRED\_INSUFFICIENT**—This code is returned when the credentials are not sufficient to validate the user.

**PAM\_AUTH\_ERR**—This code is returned when an unexpected error occurs.

16

**PAM\_FATAL**—This code is returned if there is a fatal error in the remote security server.

```
abstract protected Thread getNewInstance(String pamh, PAMDataShare dataShare)
```

This method is called to thread out when multiple users are using a Proxy Security Server and to perform conversation back to the user.

```
public int getImplementedFunctions( )
```

This method should return the methods implemented by the Proxy Security Server Classes. By default, all Proxy Security Server Classes implement only authentication. A developer does not need to modify or overwrite this method if the Proxy Security Server Class only performs authentication with no other actions. If the Proxy Security Server Class may change authentication profile and authorizations on a remote security server, then the Proxy Security Server communicates the methods that it can handle. The methods that may be handled are specified as return values set to a combination of codes shown below. To specify more than one code, append the codes together delimited by a '|'.

**F\_CHAUTHTOK**—This code is returned if the Proxy Security Server allows passwords to be changed on a remote security server. That means that Authentication and Authorization Module 114 is free to call pam\_chauthtok and this method should already be overwritten. Please refer to the pam\_chauthtok method.

**F\_GETPRIVILEGES**—This code is returned if the Proxy Security Server allows authorization to be changed on a remote security server. This means that Authentication and Authorization Module 114 is free to call pam\_get\_privileges and this method should already be overwritten. Please refer to the pam\_get\_privileges method.

**F\_GETPRIMARYPRIVILEGES**—This code is returned if the Proxy Security Server Classes support primary privileges. What this means is that Information Access System 100 identifies primary privileges to be like a user type within the Information Access System 100. The pam\_get\_primary\_privileges method should be implemented because Authentication and Authorization Module 114 can call it after authentication.

**F\_SETPROFILES**—This code is returned if the Proxy Security Server supports property setting to the external system. This means that attributes can be changed in the source system and this method is in charge of setting those attributes. Also, this method is in charge of displaying the current values or options to the users so that they are aware of the changes.

```
public int pam_chauthtok(String pamh) throws PAMHandleException
```

This method is called by Authentication and Authorization Module 114 when Proxy Security Servers 130 allows changing passwords on the system. The pam\_chauthtok implements, for example, challenging the user and changing the user's password in a Remote Security Server 140.

```
public String[ ] pam_get_privileges(String pamh) throws PAMHandleException
```

This method is called by Authentication and Authorization Module 114 when a Proxy Security Server allows authorizations to be changed on a Remote Security Server 140. The pam\_get\_privileges is in charge of returning a list of privileges that the user has in the source system. For example, Windows NT users have groups associated with them; so, the Proxy Security Server Class can return the list of groups associated with the user. These privileges are then passed to Authentication and Authorization Module 114, which is responsible for converting the privileges to access roles.

17

public String[] pam\_get\_primary\_privileges(String pamh) throws PAMHandleException

This method is called by Authentication and Authorization Module 114 when remote authorization is enabled. The pam\_get\_primary\_privileges returns a list of main privileges that the user has on the Remote Security Server 140. For example, under Windows NT Domain Authentication, users have a primary group associated with them. The Proxy Security Server can return the group associated with the user. These primary privileges are then passed to Authentication and Authorization Module 114 and Authentication and Authorization Module 114 converts the privileges into user types.

public int pam\_set\_profile (String pamh) throws PAMHandleException

This method is called by Authentication and Authorization Module 114 when user profile information may be changed on a Remote Security Servers 140. The pam\_set\_profile changes specific property or attribute information in a Remote Security Servers 140 source system. For example, a Proxy Security Server allows a user to change their name, address, and marital status stored in a user profile on a Remote Security Servers 140. This method needs to perform the conversation to the user and ask the specific information from the user himself. This method should return one of the following return codes:

PAM\_SUCCESS—This code is returned when the authentication process was successful.

PAM\_ACCT\_EXPIRED—This code is returned when the user account is inactive.

PAM\_USER\_UNKNOWN—This code is returned when the user is a unknown user or the userid, password combination is not valid.

PAM\_AUTH\_ERR—This code is returned when an unexpected error occurs.

PAM\_FATAL—This code is returned if there is a fatal error in the external system. Either the dynamic library did not load correctly or can not communicate with the external system with the given configuration.

public String get\_pam\_unique\_user(String pamh) throws PAMHandleException

This method is called by Authentication and Authorization Module 114 after authenticating the user. This is the unique userid for the user for a specific Proxy Security Server Class. The need for this method stems from the Proxy Security Server's ability to handle different users with the same userid that reside in different groupings. For example, Windows NT Domain Authentication differentiates users with the same userid from different domains by pre-pending the domain name followed by a back slash "\". The default behavior for this method is to return the userid that was entered by the user. Therefore, this method does not need to be changed if there is no differentiation between users in different groupings.

public void pam\_finish\_auth(String pamh, String gaUserLogin) throws PAMHandleException

This method allows a Proxy Security Server to perform additional tasks after a user is authenticated. For example, the Proxy Security Server may save some information in a database or flat file. The string gaUserLogin is data identifying the user in issue. The default behavior is to do nothing. This method is only called after user login was successful and the user was authenticated.

public String[] get\_associated\_pam(String pamh, String gaUserLogin) throws PAMHandleException

This method is called after a successful self-registration and the Proxy Security Server decides to make the user login

18

to another Proxy Security Server that is not the one that the current Proxy Security Server supports. For instance, a self-registration for userid/password can request Authentication and Authorization Module 114 to associate the user with LDAP [what is ldap?] because the Proxy Security Server might have created the LDAP account. Authentication and Authorization Module 114 calls pam\_finish\_auth right after the account is created, but before the association of the authentication method.

abstract public String pam\_version()

This method needs to be implemented by a Proxy Security Server Subclass to return its version.

abstract public String pam\_auth()

This method needs to be implemented by a Proxy Security Server Subclass to return the Proxy Security Server Subclass author.

public anyPAAM() & public anyPAAM(String name, String authSourceName)

There are two constructors for any Proxy Security Server Subclass that are used to initialize the base class. These constructor can also initialize private member variables after calling the base class.

public boolean setConfiguration(String array[])

This method is called by Authentication and Authorization Module 114 to set the configuration for a Proxy Security Server Class. Extra functionality can be added to this method after calling the base class.

## APPENDIX II

Methods Used to Access Services Provided by Authentication and Authorization Module

protected String pam\_get\_configuration(String key)

This method is used to get the data about the configuration of a Proxy Security Server. You can get any configuration parameter by just giving the name of the configuration parameter. The parameters values are specified by the parameter 'config' in Proxy Configuration Data 113.

protected String pam\_getenv(String pamh, String name) throws PAMHandleException

This method is used to get any environment variables from the Authentication and Authorization Module 114. Valid parameter names may be are:

REMOTE\_ADDR—The IP address from where the user is accessing the information access system.

SERVER\_NAME—The name of the server hosting the Web Server communicating with the users browser. [Note to inventor: Please explain what this parameter is].

SERVER\_PORT—The port number where the Web Server is listening to HTTP requests.

HTTP\_USER\_AGENT—The name of the user's browser (Netscape, MSIE, etc.).

HTTP\_REFERER—The referrer page. The page that sent the user to the current script.

REMOTE\_USER—The userid of the user trying being authenticated.

protected Hashtable pam\_getenvlist(String pamh) throws PAMHandleException

This method is used to get the list of all the environment variables that are sent by the Information Access System 100. This can be used to find out what variables are currently be sent by the Information Access System 100.

public String pam\_get\_item(String pamh, int itemType) throws PAMHandleException, PAMSymbolException

This method is used to get a specific item of information related to a session. The possible values for itemType, and

the corresponding information that is returned when the itemType is set, are shown below.

PAMDataShare.PAM\_USER—The userid.

PAMDataShare.PAM\_AUTHTOK—The user's password.

PAMDataShare.PAM\_OLDAUTHTOK—The user's old password.

PAMDataShare.PAM\_AUTH\_PURPOSE—The login purpose. This item type is used when the Proxy Security Server Class is configured to perform different types authentication. The purposes can be:

1. AUTH\_PURPOSE—Authentication purpose.
2. SELFREG\_PURPOSE—Self-registration purpose.
3. OTHERSELFREG\_PURPOSE—Adding authentication method purpose.

protected String pam\_get\_user(String pamh) throws PAMHandleException

This method is used to get the userid of the user. This function is exactly the same as calling pam\_get\_item with the PAM\_USER as the item type.

protected String[] pam\_conv(String pamh, int msgType[], String msg[])

This method is used by a Proxy Security Server to communicate back to the user. This is the only way in which a Proxy Security Server can ask specific questions to the user. This method requires the message type and the actual message. The message types allowed are:

PAM\_PROMPT\_ECHO\_OFF—The message type causes the user input not to be displayed. That means the user will not see what he/she is typing. The text that is placed as the label will be translated with the message itself.

PAM\_PROMPT\_ECHO\_ON—The message type is a normal input text. The label will be translated with the message of the same array index.

PAM\_PROMPT\_RW—This message type will display a translated label with a non-translated field that is read and write. This means that the user will see the current value for the label and change it.

PAM\_PROMPT\_R—This message type will display a translated label with a non-translated field that is read only. This means that the user will only see the current value for the label.

PAM\_ERROR\_MSG—This is an error message that will be displayed in the browser the user is using. The error message is a label that will be translated.

PAM\_TEXT\_INFO—This is a textual information that will be displayed to the user. The message is actually a label that will be translated.

PAM\_NOXLATE\_TEXT—This message type is the same as PAM\_TEXT\_INFO, but the message is not translated.

PAM\_CHOICE—The message type is used to generate selections that may be selected by the user. Both the label and the choices will be translated. The delimiter between the label and the choice is a colon (":").

PAM\_NOXLATE\_CHOICE—This message type is the same as PAM\_CHOICE with the only difference that the choices will not be translated.

protected void set\_item\_object(String pamh, Object obj) throws PAMHandleException

This method is used to set an object that will be later used by the Proxy Security Server Classes. This is very useful for storing information when doing self-registration because some methods are called at the end of the self-registration

process to perform any action needed by the Proxy Security Server. The best way to use this method is to store a hashtable with information and then retrieve them. A sample usage is to store information that is used in the authentication method and then retrieve them in the pam\_finish\_auth method. The pam\_finish\_auth is called after a user self registers.

protected Object get\_item\_object(String pamh) throws PAMHandleException

This method is used to get an object that set from a previous called method. This method can be used in the pam\_finish\_auth to retrieve the information that was set previously.

What is claimed is:

1. A method of selectively authenticating and authorizing a client seeking access to one or more networked computer systems that are protected by an access control system, the method comprising the computer-implemented steps of:

receiving a request of a client to access one of the computer systems;

requesting a proxy security server to authenticate the client using information identifying the client;

receiving an authorization of the client from the proxy security server based on authentication results received from a remote security server that is coupled to the proxy security server;

establishing access rights of the client, based on one or more access information records received from the remote security server through the proxy security server, for use by the access control system in determining whether to grant the client access to the computer systems.

2. A method as recited in claim 1, wherein requesting a proxy security server to authenticate the client using information identifying the client comprises requesting a proxy security server, selected from among a plurality of proxy security servers each of which is coupled to and associated with a different remote security server, to authenticate the client using information identifying the client.

3. A method as recited in claim 2, wherein receiving an authorization of the client comprises receiving an authorization of the client from the selected proxy security server based on authentication results received from the remote security server that is coupled to and associated with the selected proxy security server.

4. A method as recited in claim 1, wherein requesting a proxy security server to authenticate the client comprises invoking an authentication method having an implementation that is specific to the remote security server.

5. A method as recited in claim 1, further comprising modifying one or more authorizations that are stored on the remote security server based on a session identifier associated with the client.

6. A method as recited in claim 1, further comprising modifying a user profile that is associated with the client and stored on the remote security server.

7. A method as recited in claim 1, further comprising receiving information defining services that the proxy security server is capable of providing.

8. A method as recited in claim 1, further comprising self registering the client in a database of an access control system that controls access to the protected computer systems when user identification information received from the client is authenticated by the proxy security server.

9. A method as recited in claim 2, wherein establishing access rights of the client further comprises receiving one or more authorizations from remote security server through the

21

proxy security server, and converting the authorizations into access roles that are associated with the client based on a mapping that is stored in within a set of configuration information, wherein the configuration information comprises a plurality of blocks of configuration data, wherein each block of configuration data is associated with one of the proxy security servers.

10. A method as recited in claim 9, further comprising persistently storing the converted access roles in a database of an access control system that controls access to the protected computer systems.

11. A method of providing a security mechanism for one or more computer systems, the method comprising the steps:

a first server receiving a message specifying a request to register a user that is unregistered on the first server; wherein the first server is configured to receive requests to authenticate users and supply information that indicates access rights of users;

the first server causing a transmission to a second server requesting data that indicates access rights specified by the second server for the user;

wherein the second server is configured to receive requests to authenticate users and supply information that indicates access rights of users;

the first server receiving data, transmitted by the second server in response to receiving the transmission, that indicates access rights specified by the second server for the user including at least one authorization;

storing data that indicates the at least one authorization; persistently storing data in one or more access information records that indicates:

the user is registered on the first server, and whether access rights for the user should be obtained from the second server; the first server subsequently receiving a request to login the user; and

in response to receiving the request to login the user, establishing access rights based on the one or more access information records.

12. The method of claim 11, further including the step of: persistently storing data in the one or more access information records that indicates that access rights of the user should be obtained from the second server;

wherein the step of establishing access rights based on the one or more access information records includes:

examining at least a portion of the one or more access information records to determine that information about access rights of the user should be obtained from the second server; and

in response to determining that information indicating that access rights of the user should be obtained from the second server, the first server causing the second server to supply the at least one authorization.

13. The method of claim 11, wherein the step of storing data that indicates the at least one authorization includes persistently storing data in the one or more access information records that indicates the at least one authorization;

wherein the step establishing access rights based on the one or more access information records includes generating, from the one or more access information records that indicates the at least one authorization, data that establishes the authorization as an access right.

14. The method of claim 11, wherein the step of the first server causing a transmission includes transmitting to a third

22

server that is dedicated to providing the first server data specifying the access rights specified by the second server for a set of users.

15. The method of claim 12, further including the step of transmitting a message to one or more other servers that specifies a request for access rights specified by each of the one or more other servers for a set of users, and

wherein the third server and the one or more other servers communicates with the first server through a API.

16. A method of providing a security mechanism for one or more computer systems, the method comprising the steps: a first server receiving a message specifying a request to determine access rights of a user registered on the first server;

wherein the first server is configured to receive requests to authenticate users and supply information that indicates access rights of the users;

the first server causing a transmission to a second server that requests data that indicates the access rights specified by the second server for the user;

wherein the second server is configured to receive requests to authenticate users and supply information that indicates access rights of users;

the first server receiving data, transmitted by the second server in response to receiving the transmission, that indicates a first set of authorizations specified by the second server for the user;

the first server translating data that indicates the first set of access rights specified by the second server to one or more records that indicates a second set of access rights recognized by the first server; and

establishing a third set of access rights based on the one or more records.

17. The method of claim 16, further including the steps of: persistently storing data representing the second set of authorizations;

after persistently storing, the first security server subsequently receiving a request to login the user; and

in response to receiving the request to login the user, establishing access rights that include the second set of authorizations based on the persistently stored data.

18. The method of claim 17, wherein the step of the first server translating data includes generating data representing access roles that correspond to the first set of authorizations.

19. A method of providing a security mechanism for one or more computer systems, the method comprising the steps:

causing start up of a plurality of proxy servers that provide to a first server data that indicates access rights specified for users by a respective server from a second set of servers that are each configured to receive requests to authenticate users and supply information that indicates access rights of users;

the first server transmitting, to each proxy server of the plurality of proxy servers, a request for data indicating the access rights specified by the respective server from the second set of servers for a particular user by invoking a function of an application programmer interface that includes a common set of functions that: is associated with the plurality of proxy servers, and provides an interface between the first server and the second set of servers; and

in response to each proxy server of the plurality of proxy servers receiving the request for data indicating access rights of the particular user:

the each proxy server obtaining information about access rights of the particular user from a server of the set of servers, and

23

the each proxy server supplying information about access rights of the particular user to the first server.

20. The method of claim 19, wherein the step of the first server causing the start up includes instantiating each proxy server as an instantiation of a subclass of a parent class that defines application program interface.

21. The method of claim 19, further including the step of obtaining user input by performing the steps of:

a second server from the set of servers transmitting to a first proxy server from the plurality of proxy servers a user prompt message that specifies how additional user input should be elicited from a user;

the first server receiving the user prompt message; and the first server causing a user interface to obtain user input in a manner specified by the user prompt message.

22. The method of claim 21, wherein the step of obtaining user input includes obtaining user input that specifies a user profile for the user.

23. The method of claim 21, wherein the step of obtaining user input includes obtaining user input that specifies authentication input for the user.

24. The method of claim 19, further including the steps of: the first server receiving data from the first proxy server from the plurality of proxy servers that is supplying information about access rights of the particular user including data indicating that the particular user is registered on the respective server from the second set of servers; and

in response to the first server receiving the data from a first proxy server, the first server registering the particular user.

25. The method of claim 24,

wherein the method further includes the steps of presenting a user with a selection of names that each correspond to a proxy server from the plurality of proxy servers;

selecting a name corresponding to the first proxy server; transmitting a request to the proxy server to authenticate the particular user; and

wherein the data from a first proxy server was transmitted by the first proxy server in response to the request to authenticate.

26. An access security system, comprising

a first server configured to receive requests to authenticate users and supply information that indicates access rights of users;

a set of one or more servers that are each configured to receive requests to authenticate users and supply information that indicates access rights for users;

a plurality of proxy servers configured to provide to the first server data that indicates the access rights specified for users by a respective server from the set of one or more servers;

the plurality of proxy servers each configured as instantiations of a subclass belonging to a base class that defines an application program interface through which

24

the plurality of proxy servers and the first server interact to provide the first server with information that indicates access rights for users;

a topology mechanism configured to transmit to the first server information specifying which proxy server of the plurality of proxy servers are running; and

an access server configured to collect authentication input from a user attempting to log into the access security system and to transmit data representing the collected authentication input to the first server.

27. A computer-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, cause the one or more processors to selectively authenticate and authorize a client seeking access to one or more protected computer systems over a network, by:

receiving a request of a client to access one of the computer systems;

requesting a proxy security server to authenticate the client using information identifying the client;

receiving an authorization of the client from the proxy security server based on authentication results received from a remote security server that is coupled to the proxy security server;

establishing access rights of the client based on one or more access information records received from the remote security server through the proxy security server.

28. An apparatus for selectively authenticating and authorizing a client seeking access to one or more protected computer systems over a network, comprising:

a processor; and

a memory having one or more sequences of instructions stored therein which, when executed by the processor, cause the processor to carry out the computer-implemented steps of:

receiving a request of a client to access one of the computer systems;

requesting a proxy security server to authenticate the client using information identifying the client;

receiving an authorization of the client from the proxy security server based on authentication results received from a remote security server that is coupled to the proxy security server;

establishing access rights of the client based on one or more access information records received from the remote security server through the proxy security server.

29. A method as recited in claim 9, further comprising persistently storing the converted access roles in a database of an access control system that controls access to the protected computer systems, whereby authorizations managed by the remote security server are dynamically migrated to the access control system.

\* \* \* \* \*



**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**